Proceedings

39th Annual
Pacific Northwest Software Quality Conference

Held Virtually
October 11-13, 2021

This page is intentionally left blank.

Table of Contents

This page is intentionally left blank.

Forward

This is the 39th year of the Pacific Northwest Software Quality Conference (PNSQC). We think you will find PNSQC to be one of the best conferences in terms of learning state-of-the-art and practical methods for improving software quality. PNSQC 2021 continues this tradition.

Because of COVID-19, PNSQC 2021 is the second consecutive year for which the conference must be held Virtual Only, using the video conferencing tool Zoom. This year's conference featured speakers from around the globe in four tracks. During the three days of PNSQC 2021, our technical program consisted of keynote addresses, peer-reviewed papers and presentations, panel discussions, tutorials, workshops, and virtual exhibits of our sponsors. For additional details, see www.pnsqc.org.

The Proceedings for this year contain two main sections:

- Section A – Conference Papers. There are 15 peer-reviewed papers.
- Section B – Select Summaries of Conference Presentations. There are four summaries of peer-reviewed presentations. We gave authors of presentations without paper the option of submitting summaries for inclusion in the Proceedings, and four exercised this option.

We hope you find these materials informative and useful.

Best Regards,

Ying Ki Kwong, PhD
Board Member
PNSQC

This page is intentionally left blank.

Section A:

Conference Papers

This page is intentionally left blank.

Improving AppSec while building DevSecOps pipeline

Suresh Chandra Bose, Ganesh Bose

SureshChandra.GaneshBose@cognizant.com

Abstract

More than 85% of the applications from public App store like Apple Store and Google Play violate one or more of the top 10 risks and vulnerabilities identified by OWASP. That clearly shows the current state of our insecure apps and hence the importance of DevSecOps is even more prominent today with the need for transformational shift to improve the AppSec.

By integrating application security principles and practices into software development and operations, teams can deliver with more agility but at the same time not compromising application security.
The paper will articulate how to apply the DevSecOps best practices from Gartner across the different pillars of Continuous Delivery Pipeline. Threat Modeling as a service (TMaaS) is carried out to help discover the vulnerabilities and plug any gaps in security controls by identifying the threats and build the necessary protection into your DevSecOps workflows. With 60%-80% of today's typical application is open source code, the primary focus is to identify and removing Known Open-Source vulnerabilities.

The effective outcomes are measured by tracking 6 key metrics to validate if DevSecOps is successfully implemented. When done right, DevSecOps goes well beyond "shifting security left" (getting involved early) to "shifting security everywhere", ensuring application is secure in development, delivery and in production. When security is integrated in the DevOps pipeline, it comes with faster delivery and improved security posture enabling greater overall business success.

This paper will discuss real-world scenarios and answer the following questions:

- How Developers, Testers and Ops team work together to protect security?
- How can DevSecOps be adopted for Digital applications?
- How is Pen Testing different from SAST and DAST?
- What are the top 6 metrics every CISO must implement?

Biography

Suresh Chandra Bose, Ganesh Bose *is a Senior Manager - Consulting at Cognizant Business Consulting practice. Suresh is an accredited Lead Assessor from TMMi Foundation and has been in the IT Industry for more than 23 years with vast consulting experience in various industries. He has executed strategic initiatives for many Fortune 100 companies in the areas of PMO, PPM, Process Consulting, Program Management, TMMi Assessment/ Implementation, Organization Strategy, Test Consulting and CIO/Governance Dashboard/Metrics across the globe.*

Suresh holds 21 International certifications in IT and a speaker in 15+ international conferences, such as American Society for Quality (ASQ) Innovation Conference, American Software Testing Qualifications Board (ASTQB), 8.8 Computer Security Conference, DevOps Days Austin, DevOps Days Medellin, DevOps Days Rio de Janeiro, DevOps Days Tampa Bay, DevOps Days Berlin, Docker Community with JFrog, HUSTEF Hungary and the Pacific Northwest Software Quality Conference (PNSQC). Suresh has been part of the selection and review panel for a leading Software Conference.

1 Background

According to Gartner research, by 2022, 90% of all software development projects will be following DevSecOps practices (up from 40% in 2019). From the recently released 'state of the DevOps report - 2021', our IT industry needed an explicit call to action to start including security from the beginning of the software development lifecycle. For many organizations, the relationship between the security function and the design part of software development was even more distant than that between development and operations.

2 What is DevSecOps?

A simple DevSecOps definition would be having development, security and operations team working collaboratively integrating security in every phase of the software development lifecycle as depicted in figure 1. By having application security practices built into the software development lifecycle, application team can deliver at speed without compromising security.

It is a mindset and a way of working that ensures everyone is responsible for security in the organization.

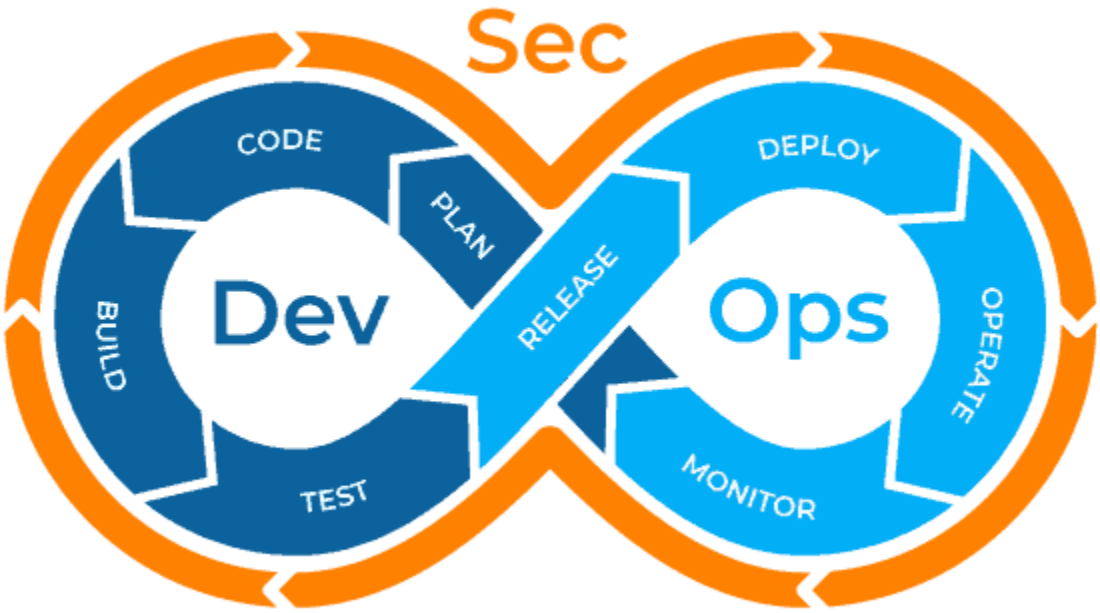


Figure 1: DevSecOps overview

3 Why DevSecOps?

The below figure 2 on the security tools usage by team across organizations is from the result of a recent survey conducted by Synopsis who is a leader in Gartner's Magic Quadrant for Application Security Testing. The results display the low usage of security tools to protect vulnerabilities. With limited focus given to the DevSecOps tool adoption, the need is huge to increase investments in various tool set to have a Secured product.

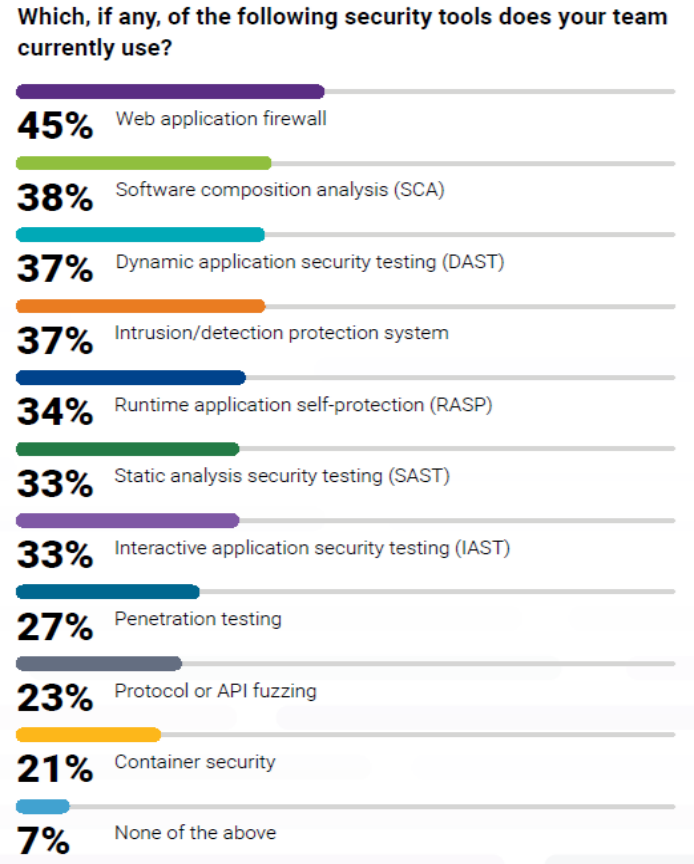


Figure 2: Synopsys Survey – DevSecOps practices and Open Source Management in 2020

4 Continuous Delivery Pipeline

The Continuous Delivery Pipeline contains four aspects as shown in figure 3 as per Scaled Agile Framework® (SAFe®) methodology. They are as follows:

- Continuous exploration
- Continuous integration
- Continuous deployment
- Release on demand

Figure 3: Continuous Delivery Pipeline © Scaled Agile, Inc.

Continuous Exploration (CE) is the initial phase of continually exploring Customer and market needs, fostering innovation and trying to build alignment on the Vision, Roadmap, and Feature set for a Solution. Continuous Integration (CI) is the process of developing, testing, integrating, and validating Features in a staging environment where they are ready for deployment and release. Continuous Deployment (CD) is the process that takes validated functionalities or features in a staging environment and deploy them to production through automated deployments, where they are ready for release. Release on Demand is the process by which functionalities or features deployed into production are released immediately or incrementally based on market needs. CALMR represents Culture, Automation, Lean flow, Measurement, and Recovery.

Let us see how security is injected into all these different aspects.

4.1 Continuous Exploration

Threat modeling is a key practice in Continuous Exploration to identify and prioritize potential threats to protect the entire system. By continuously applying threat modeling techniques, vulnerabilities and threats can be mitigated enhancing the security of the applications across the Organization.

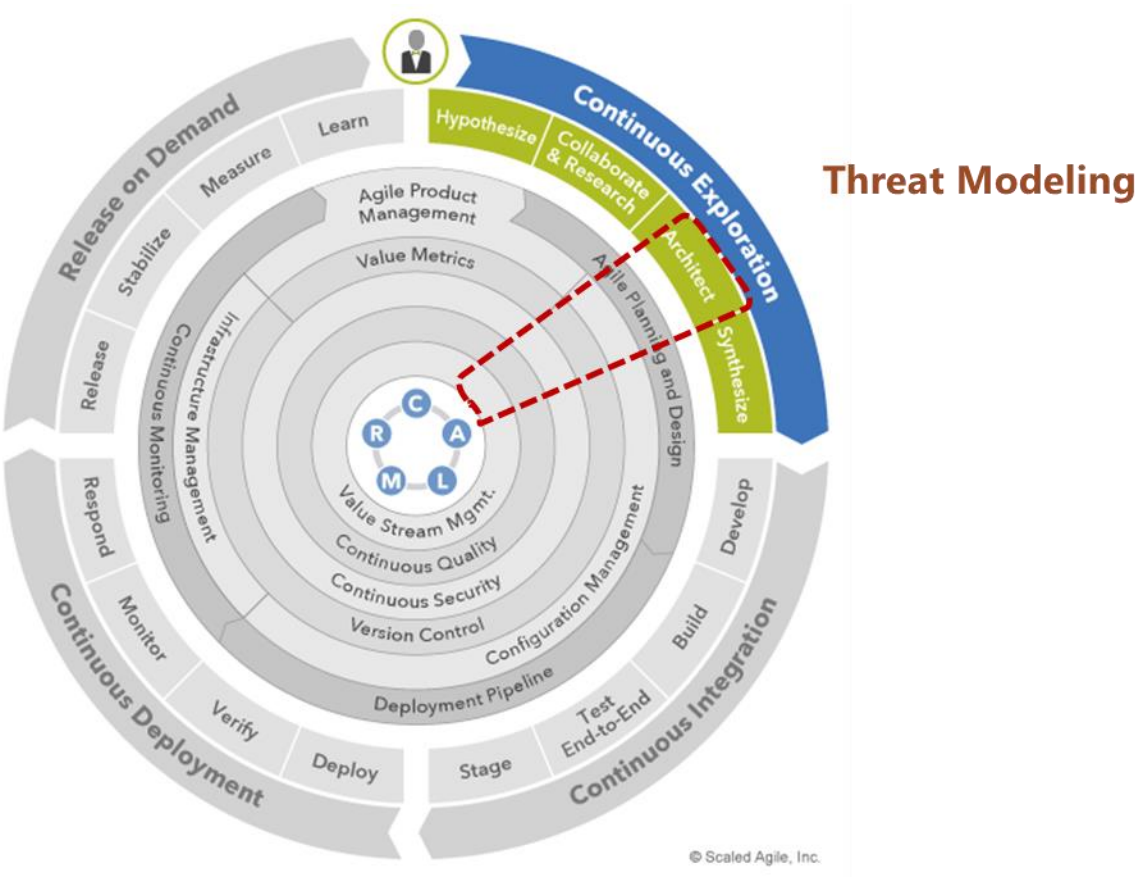


Figure 4: Security in Continuous Exploration © Scaled Agile, Inc

4.1.1 Threat Modeling as a service (TMaaS)

Threat Modeling helps in the followings ways:

- To identify the security requirements,
- To pinpoint security threats and potential vulnerabilities,
- To quantify threat and vulnerability criticality, and
- To prioritize remediation methods to mitigate

Many Organizations started using Threat Modeling as a service (TMaaS) to protect their Enterprise and remediate with end-to-end security strategy and threat intelligence methods. The strategy involves 4 key steps:

Step 1-> what are we building?

Step 2-> what can go wrong?

Step 3-> what are we doing to defend against threats?

Step 4-> have we acted on each of the previous steps?

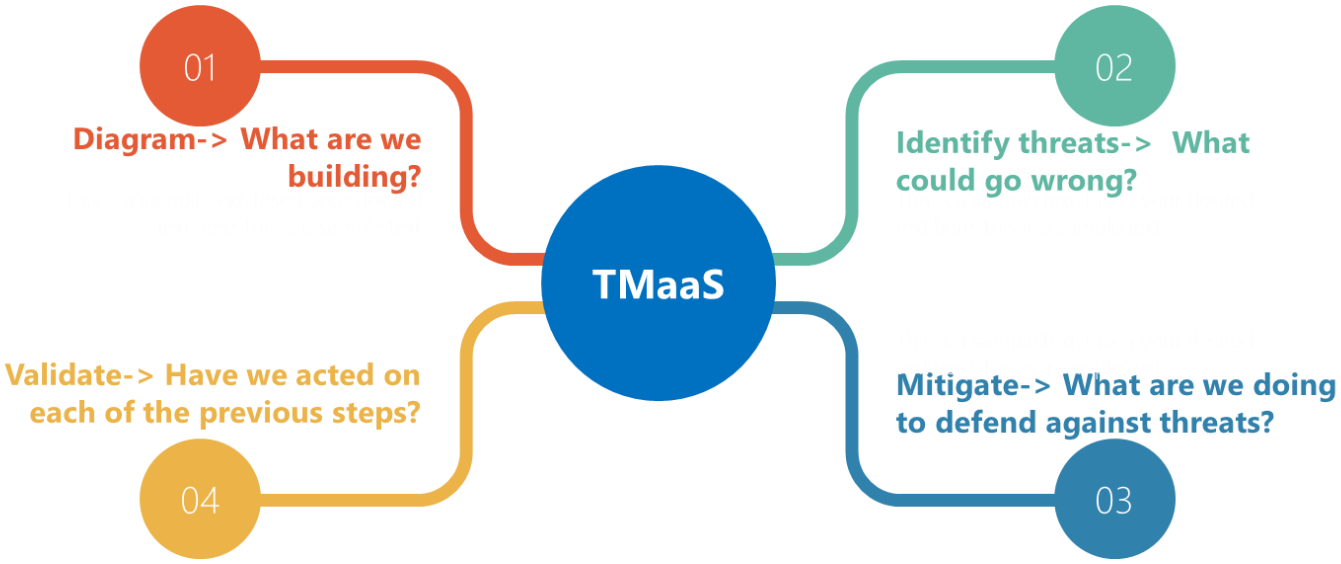


Figure 5: Threat Modeling as a service (TMaaS)

4.2 Security in Continuous Integration

As the core design and development is implemented in the Continuous Integration, there is a huge need for security measures to be built. The following are the CI practices to build security:

Security is heavily built in the Continuous Integration as the core design and development is implemented. The following are the CI practices to build security:

Security IDE plug-ins: Security is highly managed using Integrated Development Environment (IDE) plugins which help developers check for security flaws dynamically while they write their code. These plug-ins provide direct feedback while code is written. Some popular plug-ins are Eclipse, VSCode and IntelliJ.
Code reviews: A secure code review identify security threats, vulnerabilities and weaknesses that might go undetected if not reviewed. Code review can be manual and/or automated review of an application's source code done by the developers to identify security-related weaknesses (flaws) in the code component.
Pair work: Security subject matter experts are paired temporarily with developers providing real-time feedback directly during design or coding. Pair work also enhances the skill level of the development team on the security practices.
Static code analysis: It is called SAST or Static Application Security Testing, also known as, “white box testing”. The source code and code components are analyzed using SAST tools to identify sources of vulnerabilities. The Static analysis tools can detect 50% of existing security vulnerabilities based on the analysis report and is an important element of DevSecOps. Veracode, Synopsys, Checkmarx, and MicroFocus are the leaders in “The Forrester Wave Static Application Security Testing report” published in Q1 2021.
Third party scans: This is similar to static code analysis for third part code or code component, which must be scanned to identify vulnerabilities.
Fuzz testing is an automated quality assurance technique where you provide invalid, unexpected, or random data in an attempt to make it crash.
Code signing: It is a digital signature added to the application to validate that the code is not tampered enhancing the user confidence and trust.

Infrastructure scans are performed to assess the security level of your infrastructure, which includes application servers, databases, ports, as well as outdated components. Infrastructure-as-Code (IaC) scans are also performed on the infrastructure. Terraform, Ansible, AWS CloudFormation, Puppet and Chef are prominent IaC tools.
Malware scans should be used for SaaS infrastructure and before packaging any component.
Dynamic scans: It is called as DAST **or Dynamic Application Security Testing,** also known as “black box” testing, can find security vulnerabilities in a running application. Unlike SAST, DAST do not have access to the source code. Threats missed by Static application security testing are captured by dynamic scans.

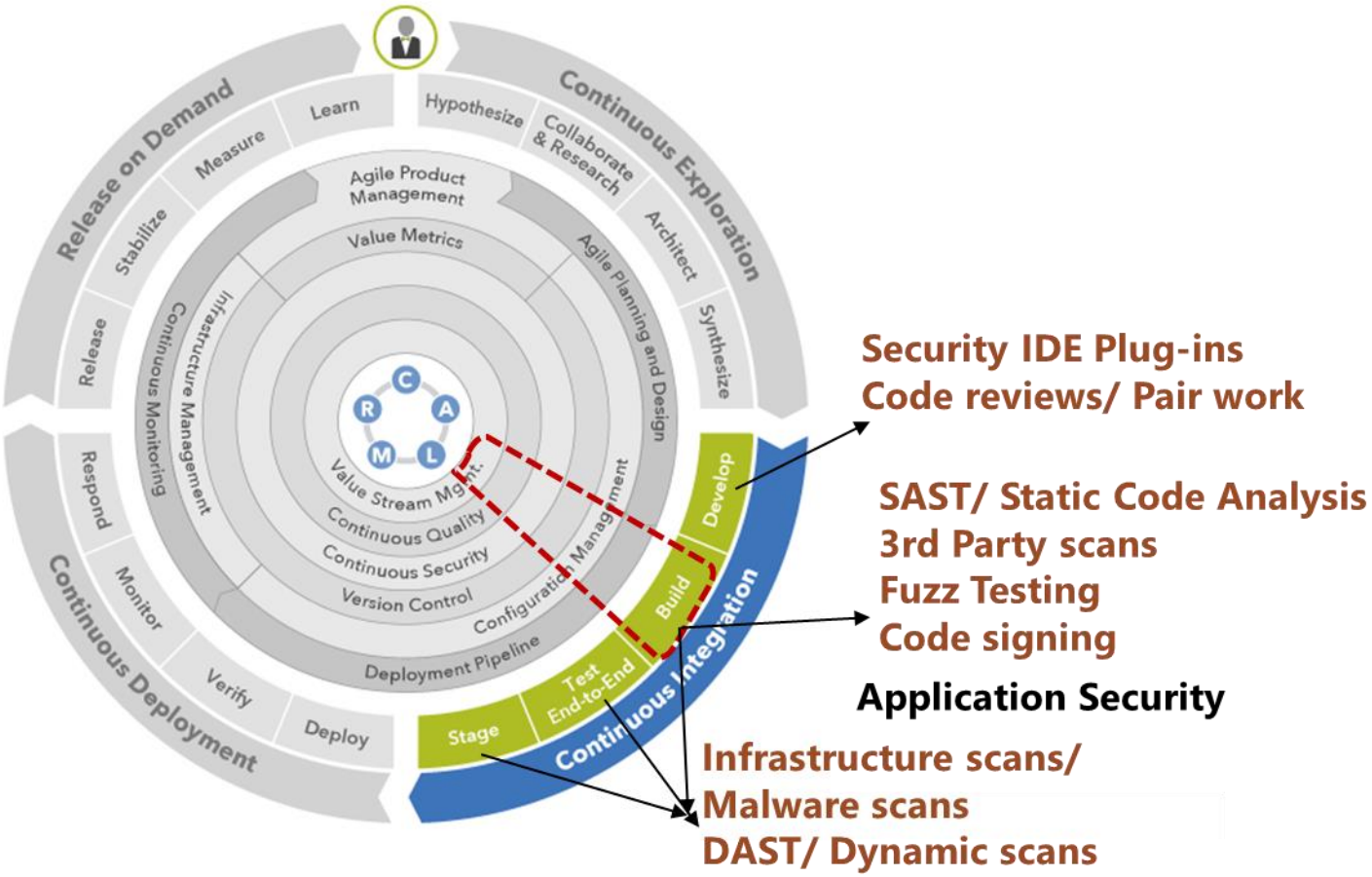


Figure 6: Security in Continuous Integration © Scaled Agile, Inc

4.3 Security in Continuous Deployment

Pen Testing or Penetration Testing is a key security practice in Continuous Deployment. It is also called ethical hacking.
The pen testing security experts conduct a series of simulated hacking or attack against the network using different methods. The different types of penetration tests include network services, applications, client side, wireless, social engineering, and physical. A vulnerability scan (SAST or DAST) is automated, while a penetration test is mostly a manual test similar to a product risk assessment performed by a security professional. Qualys and Netsparker is used widely for vulnerability scans.

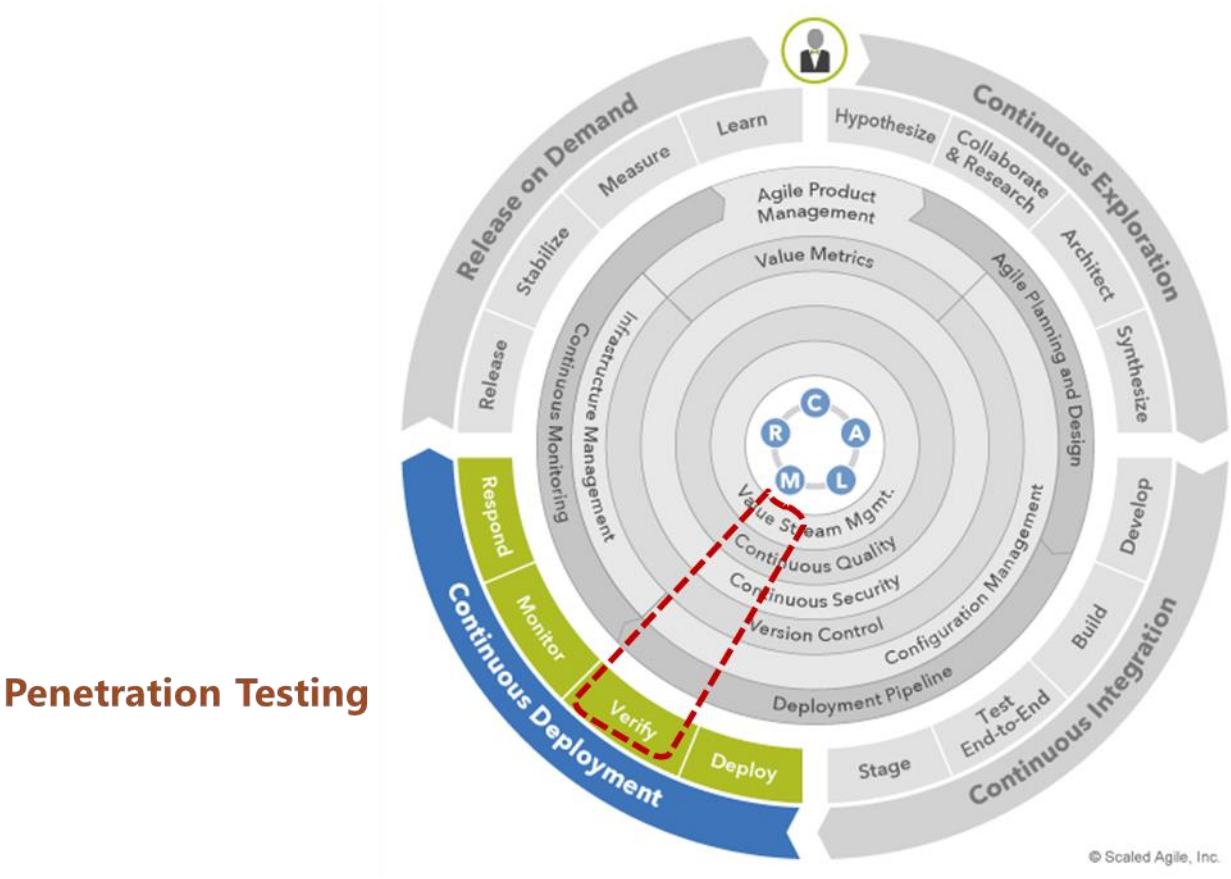


Figure 7: Security in Continuous Deployment © Scaled Agile, Inc

4.4 Security in Release on Demand

Continuous security monitoring (CSM) is a threat intelligence approach that automates the monitoring of information security controls, vulnerabilities, and other potential threats. CSM empowers companies to oversee their IT assets, both in cloud and on premise.
This is done by a security information and event management (SIEM) system that gives organizations with next-generation detection, analytics and response providing real-time analysis of security alerts generated by applications and network hardware. With the data collected, security team can take required actions and generate applicable security compliance and audit data. The leaders of SIEM from Gartner's 2021 magic quadrant are Exabeam, IBM, Splunk, Securonix, Rapid7 and LogRhythm.
The security response team or CSIRT (Computer Security Incident Response Team) provides information about newly discovered vulnerabilities and feeds this information back to the development and operations teams. CSIRT respond to computer security incidents quickly and efficiently, thus regaining control and minimizing damage. Security Bulletins are used to notify customers about the vulnerabilities for quick remediation.

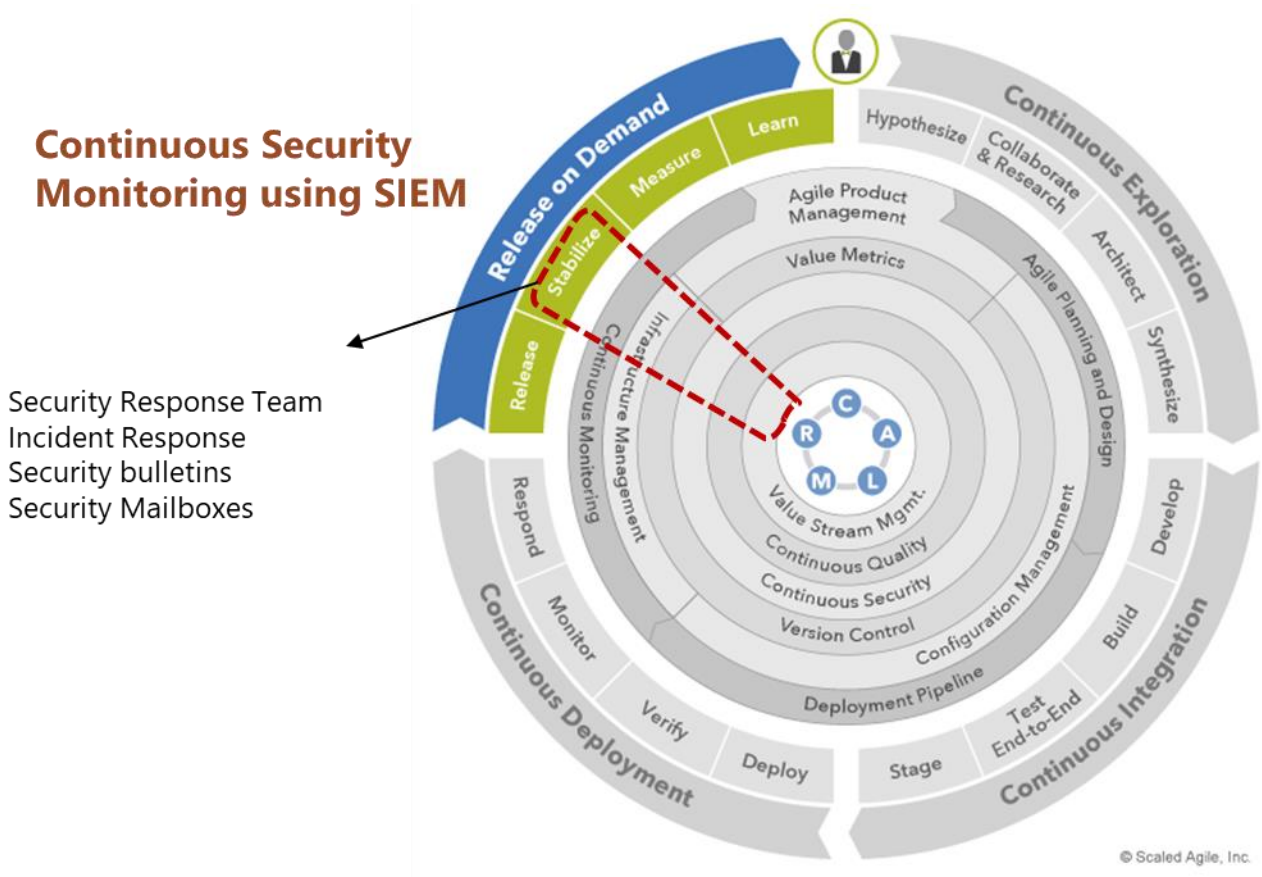


Figure 8: Security in Release on Demand © Scaled Agile, Inc

5 Key Best practices from Gartner and others leading experts

Organizations face more challenges when transforming to DevSecOps, and they can be addressed by employing these DevSecOps best practices. There are many practices adopted by security professionals in the DevSecOps adoption but here are just a few key best practices from Gartner and other leading experts for organizations seeking to run the DevSecOps practices smoothly:

- Automation is good
- Shift Left for efficiency
- Adopt a Security Champion
- Training developers on security
- Carry out threat modeling
- Implement Strong Version Control
- Focus on Known Open-Source Vulnerabilities

6 DevSecOps Metrics

There are many possible measures to track the success of DevSecOps implementation. The decision of which metrics to track is largely based on business need and compliance requirements. The following are key metrics for every CISO (Chief Information Security Officer) to ensure if the DevSecOps practices are effectively working in your Organization:

- Reduction in Total Security Tickets opened
- Reduced Time-to-Deploy
- Discovery of Preproduction Vulnerabilities
- Reduced Time-to-Remediate
- Percentage of Security Audits Passed
- Reducing Failed Security Tests

7 Summary

DevSecOps transforms the way organizations manage security. Integrating security into DevOps to deliver DevSecOps demands changed mindsets, processes and technologies. The most important and obvious benefit of a DevSecOps approach is that organizations can improve your overall security posture with faster delivery. Identifying vulnerabilities and bugs before deployment results in an exponential reduction in risk and operational cost enabling greater overall business success. DevSecOps also ensures meeting compliance with industry-standard regulations.
When done right, DevSecOps goes well beyond “shifting security left” to “shifting security everywhere.” It ensures software is secure in development, delivery and in production thereby improving overall AppSec in entire DevSecOps pipeline.

8 References

Kindle Edition: Continuous Testing for DevOps Professionals: A Practical Guide From Industry Experts by Eran Kinsbruner
https://www.amazon.com/Continuous-Testing-DevOps-Professionals-Practical-ebook-dp-B07H8PH7VB/dp/B07H8PH7VB/

Survey Report: DevSecOps Practices and Open Source Management in 2020 by the Synopsys Cybersecurity Research Center (CyRC) and Censuswide, August 2020
https://www.synopsys.com/software-integrity/resources/analyst-reports/devsecops-practices-open-source-management.html

Market Research Analyst Report: State of DevOps report, 2021
https://puppet.com/resources/report/2021-state-of-devops-report/

Analysis report: "Effect of static analysis tools on software security: preliminary investigation" (PDF). Okun, V.; Guthrie, W. F.; Gaucher, H.; Black, P. E. (October 2007). *Proceedings of the 2007 ACM Workshop on Quality of Protection. ACM: 1–5.* doi:10.1145/1314257.1314260. S2CID 6663970

Web Sites: Scaled Agile Framework, version 5.0 Continuous Delivery Pipeline
https://www.scaledagileframework.com/

Market Research Analyst Report: Forrester Wave™: Static Application Security Testing, Q1 2021
https://www.synopsys.com/software-integrity/resources/analyst-reports/forrester-wave-sast.html

Web Sites: Gartner- 12 Things to Get Right for Successful DevSecOps, Published 19 December 2019 - ID G00450792
https://www.gartner.com/en/documents/3978490/12-things-to-get-right-for-successful-devsecops

Market Research Analyst Report: Gartner's Magic Quadrant for Application Security Testing, 2021
https://www.synopsys.com/software-integrity/resources/analyst-reports/gartner-magic-quadrant-appsec.html

Market Research Analyst Report: Gartner's Magic Quadrant for for Security Information and Event Management (SIEM), 2021
https://www.splunk.com/en_us/form/gartner-siem-magic-quadrant.html

Test Optimization Through Risk-Based Validation Approach

Felix Eu
Intel IOTG Software Quality Team
felix.eu@intel.com

Tan Chin Pei
Intel IOTG Software Quality Team
chin.pei.tan@intel.com

Abstract

In real software's project day-to-day context, there are so many validation approaches available in the market and each organization may adopt the approach that is most relevant and suitable to its business and industry. There are always the opportunities to customize the current validation process to meet the specific organization's goal. The risk-based validation approach is useful in a situation where multiple software program releases happen and share the same validation resources but with increasing validation scope. It is intended to assist the validation team in a large organization to use the risk factors as an input, prioritize the validation tasks based on specific conditions and quality management output analysis. In contrast to the traditional software regression validation approaches which generally test everything when there is a code change, risk-based validation helps optimize a test strategy that will maintain the efficiency of the test coverage and ensure the right validation scope is implemented.

Biography

Felix Eu is a Software Quality Engineer at Intel Corporation based in Penang, Malaysia and is responsible for software release qualification and process improvement. Prior to this role, he was a Software Test Engineer and Test Project Lead at Motorola Solutions responsible for two-way radio software related functional testing, test planning, defect prediction, escaped defect analysis and test project management. He was a Supplier Software Development Engineer in the same organization and responsible for supplier engagement in software development and process improvement. In these roles, his efforts focus on software product quality and business process improvement. He has held the Digital Six Sigma Green Badge since 2019 and holds a Degree in Computer Science from University of Bolton in the UK.

Chin Pei has 16 years` experience in software engineering. She worked as a validation engineer, software development engineer and automation engineer in her previous company. Her extensive experiences in different software roles have helped her to contribute to the software product development, quality, and processes in Intel. Currently, she is a passionate Quality Engineer who is responsible for ensuring that software meets Intel's software quality release criteria (including areas like legal compliance and security compliance). She provides software process assessment, trainings, consultation, and guidance to the teams across different functional and regions. She has submitted automated and Lean projects which have greatly helped the validation projects to improve ROI, reduce resources and cost.

1 Introduction

In this introduction, we discuss what testing is and the important of selecting testing techniques that determine which validation approach is best suited to the organization. Subsequently we introduce the risk-based validation approach and the benefits of using it.

In Section 2, we discuss the software validation transformation, and how the organization can transform from the traditional software testing approach to the customized risk-based validation approach. We cover various transformation factors which include risk analysis, defect prediction model, project monitoring and control plan and building up subject matter experts (SME). Section 3 builds on Section 2 and provides examples of source of impact analysis.

Section 4 describes the risk management process and the various ways of leveraging the product risks, summarizing the test prioritization based on business risk and typical defect detection trend. In section 5, we elaborate on the mapping of the validation tasks to the original impact analysis summary provided by software developers.

Section 6 describes our experiments and results obtained when compared to full regression testing. This section also covers the gap analysis and maintenance process with an end target of improving the current risk-based approach. In section 7, we provided the implementation results of risk-based validation approach. Finally, in Section 8, we summarize and provide directions for future work and areas to research.

1.1 What is testing?

What is testing or validation? According to *ISTQB Certified Tester Foundation Level Syllabus, 1.1. What is Testing?* [5] “Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.” Testing is the process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

1.2 Testing Techniques Selection

Selecting an appropriate testing technique has been the key element of fulfilling the testing goal. It is obvious that appropriate testing techniques should be used in conjunction with the process of dealing with testing constraints and risk analysis. In general, both risk analysis and test constraints play important parts to ensure the right testing technique is selected. We need to prepare an appropriate methodology, putting in all the test constraints and risk factors, and perform simple evaluation to check if each of the available testing technique works and can achieve the testing goal. This is to avoid time wasted due to inappropriate testing technique being selected.

Let’s review the examples of test constraints in Figure 1. To achieve effective testing, we should test everything we can to obtain full test coverage so that we can find as many defects as possible. In contrast, to be efficient we should minimize the testing time and only perform certain tests, and we might not be able to find as many defects as possible. As you can see in both scenarios, the test constraint prevents us to achieve the efficient and effective testing.

Figure 1 Test Constraint

Figure 2 represents a high-level fundamental view on risk analysis in conjunction with the testing goal and proposed testing technique. The idea is to further drill down the testing scope by evaluating the possible risks by asking a list of right questions like for example “If I only have limited time/resources to perform the validation”, “Why I should perform full validation if the code changes are on specific areas”, “Is there expectation for defects to be found if the same validation strategy is performed from one cycle to another cycle” and so on, this can narrow down the selection of testing technique.

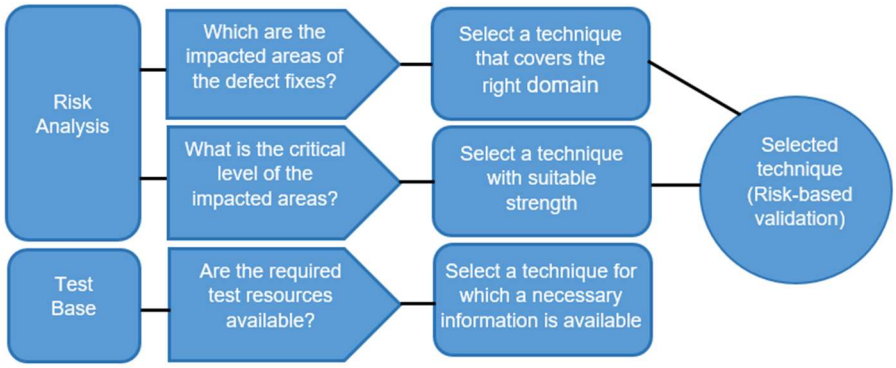


Figure 2 Risk Analysis

Apart from test constraints, other risk factors can also be measured to determine the appropriate testing technique selection for the project. In fact, there may be a combination of different test selection techniques in a single project. The depth and coverage of each of the testing techniques may vary from one to another. For instance, during the initial project execution phase, we may want to take fewer risks and perform full regression testing to ensure all the newly developed code and functions have been thoroughly validated. Towards the end of the project when time constraints are more critical, we may want to change the test strategy to reduce the test coverage by taking more risks which only specific validation will be performed based on risk assessment. Some will perform sanity test or smoke test just to ensure the basic functionalities are working accordingly.

So, if both the test constraint and risk analysis are considered in the evaluation of testing technique selection, the chances to choose the right testing technique are greatly improved.

1.3 Risk-Based Validation

The testing technique we’ll be examining here is the risk-based validation approach. In a critical situation when a software release timeline has been promised to the customer, it is crucial to ensure the entire software code is thoroughly validated. However, it is time consuming and not feasible to conduct a full regression testing whenever there’s a small change in code to ensure functional integrity of the software. Even with major code changes, not all code is updated due to the new implementation. The question that always been asked is how these code changes can be validated and what kind of validation strategy can be used to achieve the maximum validation coverage of the code changes. Risk-based validation is one of the testing approaches adopted for a project which utilizes the impact analysis done on software code changes, prioritizing tasks, and taking informed risks to perform the validation. A software development

team must provide a summary of software code changes in the format of impact analysis data based on the source code delta from previous software release cycle**.** In fact, they know very well which codes they have changed. The impact can be categorized as low, medium, high or any numbering symbol and presented in a table format which describes the relation of the software code changes between the software feature and specific function/area in a project (see Table 1). The output from the impact assessment shall be discussed in the project meeting, clearly communicated and aligned with the software validation team to determine the risk-based validation strategy used for test optimization. In addition to the software code changes, incorporating defect prediction outcome into the impact analysis will greatly enhance the test optimization through risk-based validation approach. This risk-based approach can be used whenever there is a limitation or constraint on time, cost and resources of a project and whenever there is a need to optimize the test resources. Nevertheless, the risks taken should be balanced with the Return of Investment (ROI) gained in a project.

The risk-based validation approach mainly focuses on the considerations below:

- Specific area of the product where there is a high impact on business due to failure or high likelihood of failure in the production.
- Early defects detection and allowing a team of software developers to fix them as early as possible
- Quality control to improve on 'end to end customer experience'**.**

2 Software Validation Transformation

The world of software validation has rapidly changed, and the trend indicates that it is transforming from the traditional validation to a different kind of more cost-effective and adaptable validation approaches including risk-based validation. The maturity of the validation process in an organization is the key determination for such transformation and can be measured and represented through CMMI levels. Organizations with higher maturity level tend to have focus on their process improvement efforts on a prioritized and manageable number of practice areas. The performance and process improvement achievements increase capability to level up their software validation practices. Considering the availability of the testing constraints and the process improvements in place, many large organizations have taken the precedent for the transformation in software validation approach.

The process for this transformation may include but not be limited to the following:

- Risk Analysis:
  - Risk identification and assessment
  - Risk control and mitigation
  - Risk communication and action
- Identify a defect prediction model
- Develop or enhance a monitoring and control plan
- Build up Subject Matter Experts (SME)

2.1 Risk Analysis

According to the Oxford English Dictionary, risk is "the possibility of something bad happening at some time in the future; a situation that could be dangerous or have a bad result." In software validation, risk analysis is an approach to identify and analyze risks, implement a control and mitigation plan, and measure how the risks may create threats to the functionality, stability, security, or performance of the application. The result of the risk analysis will be communicated to the relevant party and the recommended action plan will be taken to avoid any factors that can bring negative impact to the business or project.

Figure 3 shows the Risk analysis activity model. This model is taken from Karolak's book "Software Engineering Risk Management", 1996 [6] with additional comments (in blue oval).

In general, risk analysis process happened throughout different phases in software development lifecycle. At the beginning of the project execution, team is asked to perform risk identification. This covered a wide area of development and validation activities from where potential risks are likely to happen with reference to the past projects. In validation, test conditions will be the focus and high-level risk assessment will be performed against all possible test conditions. Risk assessment output will be documented, and recommended risk strategy will be included in the test plan. Of course, risk assessment and risk mitigation are continuous processes throughout validation phase and will be constantly monitored to identify any new risks. For example, validation scope from the defect analyses is unable to be fulfilled due to insufficient human effort to support the large impact areas. In this context, test plan may need to be updated to align with mitigation plan. A summary risk report should be generated in a weekly basis in accordance to each of the test metrics such as validation passing rate, validation task completion rate, defect detection rate and so on. All the open risks should be revisited to determine the current occurrence probability rate and to predict any new risks.

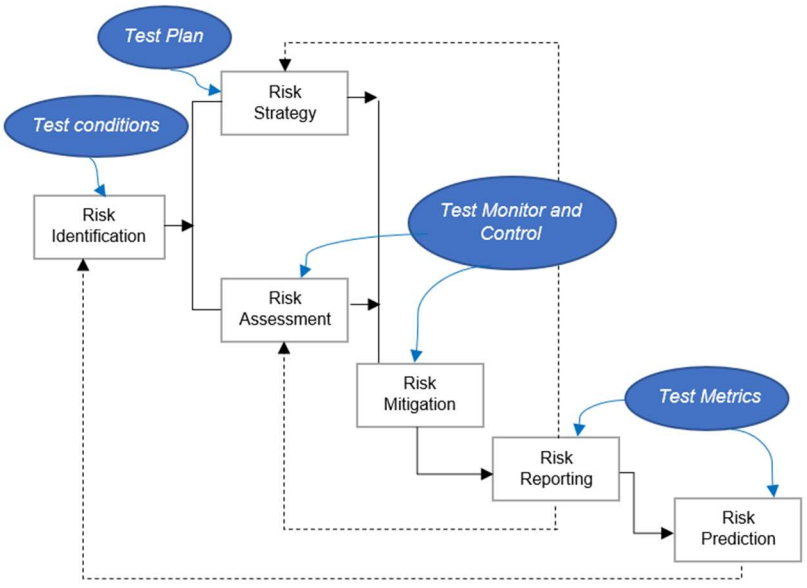


Figure 3 Risk Analysis Activity Model

2.2 Defect Prediction Model and Potential Impact

Identifying and developing a defect prediction model requires a significant effort to collect, analyze, categorize and pull the defect data from a database into a presentable chart that shows the correlation of defects uncovered between historical projects and current projects. It is one of the most powerful software metrics to describe the relationship among the complexity of the software implementation, software development time and the probability of the code errors invented by software developer.

The defect prediction outcome provides the level of confident and quality assurance of the software application under development It acts like a benchmark and quality standard to gauge the software performance by ensuring the actual defect arrival is consistently approaching the predicted number during the entire software development life cycle. To enhance further, the validation team may consider incorporating the defect prediction outcome to check whether the defect arrival rate is on-par with the defect prediction. Depending on the actual defect arrival rate, if the actual defect arrival and prediction number are not on-par, the software quality team can chip in and take immediate action to analyze why the actual defect arrival and defect prediction are misaligned. There are a lot of questions that can be asked to help to identify the root cause. If it is below prediction, software validation team can further analyze and determine if the right amount of validation has been conducted the right functional areas have been validated or the right validation strategy has been adopted. As a result, they can perform gap analysis and adjust the validation strategy to improve the test coverage.

2.3 Project Monitoring and Control Plan

According to the Project Management Body of Knowledge (PMBOK), “the Monitoring and Control Process Group consists of those processes performed to observe project execution so that potential problems can be identified in a timely manner and corrective action can be taken, when necessary, to control the execution of the project.” [12]

Organizations should have a project monitoring and control plan in place and enhanced as regular as possible to maintain their competencies in the business world. In fact, software validation also requires its dedicated monitoring and control plan. Test monitoring and test control focuses on software validation activities, efforts evaluation, risk analysis and the action plan and tracks its progress against the test metrics. Scope validation formalizes the acceptance of the final project deliverables. It ensures that the deliverables have been completed according to the plan and meet all the requirements set forth throughout the project. This is very useful when the limited test resources are available to perform risk-based validation.

The risk-based validation approach pursues a stricter monitoring and control plan as compared to full regression validation because of the test constraints. Though there is always buffer allocation during project planning, it takes more risks to execute the validation plan. The risk-based validation approach works very closely with defect prediction plan to reduce the risks. Regular audits on defect detection rate during the risk-based execution phase greatly helps to identify whether the right areas are validated, or the right SMEs are assigned to perform the validation. It helps to ensure that risks taken are reasonable and aligned to the project scope, budget, and timeline. Alerts will be promptly triggered for any unexpected outcomes that occur.

2.4 Subject Matter Expert (SME)

It is challenging to build and retain the competency of the subject matter experts. In general subject matter expertise can be built based on working experiences, talents, research, and study etc. so that SMEs can be trained to have an in-depth understanding of a job and are able to thoroughly and accurately discuss duties and responsibilities of a job, knowledge requirements, skill requirements, ability requirements, and other specific software feature knowledge.

The complexity of a software application doesn’t always come from a single feature implementation but from features from different entities, subjects and aspects - for example security, functionalities, performance, graphics, etc. Subject Matter Experts are required for various purposes during several phases of the development and validation. From software validation perspective, SMEs are needed in specific content areas to provide judgments on tests already written and validated by other SMEs. Judgments are needed for validation and setting performance standards for these tests currently being field tested.

Of course, challenges arise during the process of transformation. However, the consistency and commitment to meet the higher quality standard of software validation pays off all the efforts invested for this change.

3 Source of Impact Analysis

Impact analysis helps to identify the potential consequences of any changes made on software. While software code changes are unavoidable and these changes may come from various sources, such as business needs adjustment, new requirements, new technologies, etc., it is very likely that the changes could result in failure or become out of control. Impact analysis helps to oversee the risks of the change and the resources which we should plan forward for implementation of the change.

Defect fixing is one of the common activities in the entire software development life cycle that causes the software code to change. Prior to the commencement of defect fixing, each of the defects should be triaged and all the discussions and resolutions will be properly recorded and updated in the defect

tracking. This process will serve as proof of evidence on the general agreement of correct categorization of priority, exposure and feature of the defects. Once the defects which targeted in a predefined software release milestone are fixed, software developers are requested to enter all the impact data into the impact analysis table (Table 1) one by one and these data will be consolidated by the software program manager. Filling the impact analysis data shall be as precise as possible to avoid ambiguity. Once the software release candidate is finalized for the milestone, it will be officially communicated to the software validation team during the software handover meeting. The same process is repeated for software code changes due to new features implementation and new requirements added into the scope.

4 Risk Management and an Approach to Validation

Risk management is the process of identifying, assessing, and controlling threats to an organization's capital and earnings. Before we start on a new project, we must first perform risk identification through the risk screening process based on impact and probability. These two elements shall always be evaluated on different project characteristics. In most cases a mission critical project tends to take lesser risk as compare to normal project. The reason is risks that are characterized as both high impact and high likelihood of occurrence often cause a project to fail if it is continued despite the risks. However, it can be balance up by the potential benefits gained that taking the risks is justified.

The risks in developing the validation strategy can be reduced by taking the risk-based validation approach with proper analysis and assessment on the delta in the software code instead of repeatedly performing full regression testing. Experienced software test analysts or technical leads utilize their best knowledge to understand and interpret the data in the impact analysis table which represents the software code changes. Each of the entries in the impact analysis table will be thoroughly analyzed to capture all possible dependencies and interactions with other features. For example, software code change in Dockers containers may impact any features that have a dependency on Docker images. If the data have similar characteristics, they will be combined and consolidated. The output will be a list of tests or validation tasks that are specific to the overall software code changes. However, each test or validation task has its priority which has a correlation to the original impact when the defect is triaged.

Product risk level can be managed via risk-based validation in the following ways:

- Start the risk-based validation as early as possible to identify the correctness of the assessed impact and to determine if more validation should be performed.
- Prioritize validation in the high impact areas.
- Have mitigation and contingency plans in place to complete validation after uncovering high impact defects. For example, workarounds to continue the validation until all impact areas are covered.
- Implement measurements of how well the risk-based validation approach at finding and removing defects in critical areas.
- Conduct proactive continuous risk assessment on non-validation areas to ensure they are defect-free throughout the release cycles.

The goal of risk-based testing is not to achieve a risk-free project but to carry out the testing with best practices in risk management to achieve a project outcome that balances risks with quality, features, budget and schedule.

Figure 4 shows the risk-based validation aligns the validation activities with business priority and achieves optimal risk coverage with focused validation. The chart represents the situation where it rapidly reaches over 80%+ business risk coverage, using only 20% of your test effort [3]. Risk Coverage Optimization shifts the focus from test coverage to risk coverage.

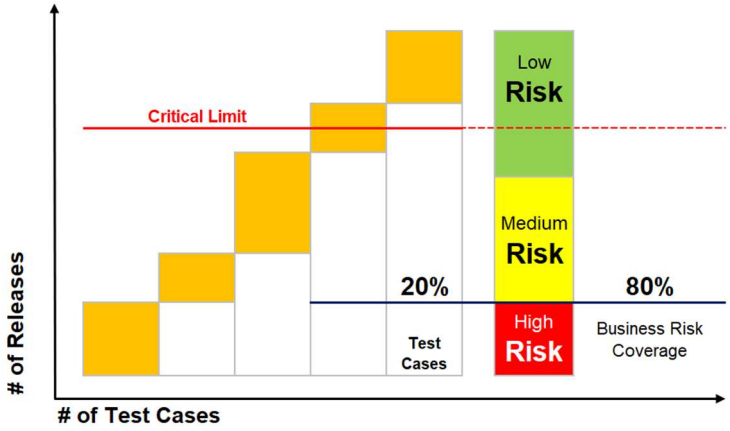


Figure 4: Test Coverage vs Risk Coverage

Figures 5 denotes a healthy defect detection trend against test hours. Though there are a number of factors that could influence the defect arrival trend, a typical defect discovery trend normally has a sharp increase during the beginning of the validation process and and then decreases after certain period of time. We must grab the opportunity and keep the right validation to find as many defects as we can during the golden hours. When the curve flattens, it is not worth to continue the validation process as the defect density is almost at its maximum limit. Though regression testing may be able to uncover most of the defects eventually however it may be too late with no specific focus in validation process. Risk-based validation functions in such situations aim to flush out as fast and as many defects as possible while maximizing the test coverage.

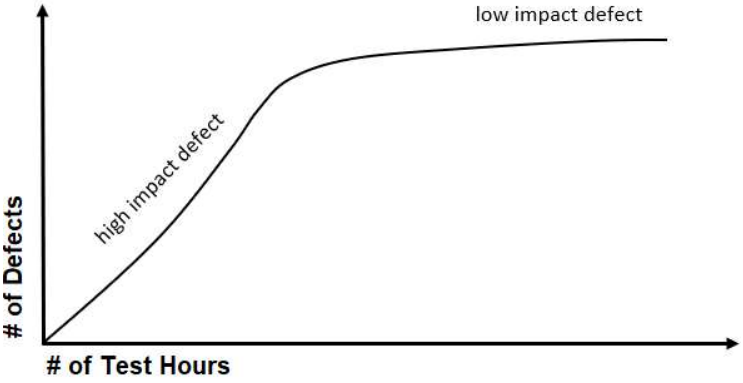


Figure 5: Number of Defects vs Test Hours

The risk-based validation approach will prioritize tests based on the impact assessment from the overall software code changes. Depending on various scenarios and situations, we can then optimize the right amount of validation to be performed on one feature over other features and we can include feature interaction tests. Both software development and validation teams play critical roles to ensure the accuracy of the software impact given (input) and the tasks defined from the impact analysis (output).

5 Validation Tasks Traceability

The efficiency and effectiveness to interpret and convert impact analysis data into a list of validation tasks is the key success of the risk-based validation approach. Defect tracking systems such as Jira, Bugzilla, HP ALM, Mantis and so on are great tools for defect management and new features implementation. Impact analysis activity requires a powerful defect management tool to improve its accuracy and completeness. The impact analyses we have discussed so far focus on the outcomes of the software

code changes where each of the outcomes are labeled with a respective “impact categorization” and precisely plotted into a two-dimensional table as shown in Table 1. Each of the entries denotes correlation between features that have code changes. In Table 1 below, the determination of the high impact was observed on Feature-1 alone so it is marked as 1 (High) in the cell that correlation between Feature-1 and Feature-1. Next, code changes happened between Feature-1 and Feature-2 however the impact is average so it is categorized as 2 (Medium). In fact, the impact analysis table aims to enhance process integrity and provide clarity to all the impacted features.

Features \ Features	Feature-1	Feature-2	Feature-3	Feature-4	Feature-5	Feature-6	Feature-7	Feature-8
Feature-1	1	2						
Feature-2								
Feature-3			1					
Feature-4	3							
Feature-5						1		
Feature-6								
Feature-7				1				2
Feature-8		3				3		

Legend:

1	High
2	Medium
3	Low

Table-1: Impact Analysis

From the impact analysis table, the intention of the software code changes has been gracefully transformed into systematic, well-defined, and testable tasks in Table 2. For example, the intersection of Feature-1 and Feature-1 has an impact of 1 (High); this mean only Feature-1 alone is impacted and requires higher validation coverage. As another example, at the intersection of Feature-1 and Feature-4 there is an impact of 3. This mean both Feature-1 and Feature-4 are impacted but the impact is 3 (Low), so only minimum validation coverage will be planned for Feature-1, Feature-4 and the interaction between Feature-1 and Feature-4. Further analysis shall also be performed to check if the impact is only happened on specific scenario, situation, or environment.

Each test or validation task must provide a traceability capability so that unrelated tests can be eliminated. Effort can also be calculated from the analyzed impact which allows software validation team to optimize test coverage with optimal effort based on the availability of test resources and prioritization of the validation tasks.

Validation Strategy for Milestone X

	Tasks	Defect ID	Priority	Total Test Cases Count	Test Cases to be tested	Effort (Staff Day)
1	Execute 20% regression test of Feature-1	12345	1	50	10	1
2	Execute 5 specific testing for Feature-4	26472	3	10	5	0.5
3	Execute 30% regression test of Feature-10	12121	2	28	8	0.8
4	Execute 8 test cases from Feature-2 on Function-1	15462	2	56	8	0.7
5	Execute feature interaction between Feature-5 and Feature-6	12358	1	NA	10	1
6	Execute stress test for Feature-3 for 10 hours	19546	1	3	3	3
					Total Effort	7

Table 2: Validation Strategy for Milestone X

Although it looks easy to conduct the impact analysis by turning them into more specific tests, obviously the efforts behind to perform such analysis is tremendous. In the previous sections of this paper, we explained that risk analysis, developing a control plan and building up subject matter experts all are key contributors to the success of such impactful activity.

6 Gap Analysis and Maintenance

Over the years, much research has been carried out to identify the root causes of software failures. One of the main reasons for such failures turned out to be poor quality assurance during the software development process. The main purpose of executing stringent quality assurance tests is to prevent the release of poor-quality software products. Small mistakes that slip through may potentially lead to large

financial losses. That's where a gap analysis comes in. It provides an objective perspective to improve processes for the future.

Gap analysis takes into consideration of different aspects surrounded quality. It is important to allocate appropriate time and depth to a review of the gap analysis. The gap analysis may identify situations where risk-based validation has never been performed or may find that risk-based validation performed is lacking in some critical details. The outcome allows us to prioritize testing efforts and detect defects early in the release cycle instead of in production. However, it is important that the effectiveness of any gap analysis lies with the technical competence of those performing the analysis. Unless those performing the analysis are technically competent and have good attention to detail traits, otherwise the effectiveness of risk-based validation that have previously been performed cannot be truly determined.

Quality reviews shall be performed on the risk-based validation approach on a regular basis to identify potential quality gaps as below:

- Impact analysis data is not available at the right time, or it is delayed
- Incorrect entries in the impact analysis table
- Misalignment between defect prediction and defect arrival trends causing misunderstandings to different parties
- Inaccurate risk assessment causing the wrong target for testing
- Ambiguous or wrong definition of the validation tasks which is unable to be matched to the impact analysis data
- Improper prioritization of the validation tasks
- Duplicate entry of validation tasks which can be combined and consolidated
- Insufficient test coverage on extremely high impact area
- Redundant test coverage on extremely low impact area

A small change in software code may have a big business impact to the organization because of the additional resources required to support it and the changes required to go through the entire software development lifecycle again. A gap analysis has a direct impact on an organization's efficiency, which in turn affects the bottom line. With the insights gleaned from the gap analysis, the respective improvements can be quickly implemented to boost performance for future activities in the following ways:

- Turn the finding into action plans which will fill in the gaps
- Develop plans to address the gaps
- Assign and track the action plans until closure of the gaps.

There are no perfect testing methodologies that can ultimately produce a defect free software product. Without exception, risk-based validation approach will still require having regular quality reviews in place to improve all the identified gaps. The aim is to maintain risk-based validation as one of the most competitive approaches to meet an organization's needs, utilizing all the possible best practices to leverage the risks, quality and all other resources. The best way to create high-quality software product is to implement effective quality management that provides tools and methodologies for building defect free software products.

7 Implementation Results

We carried out measurements between the traditional full regression testing and new risk-based validation approach on different aspects such as staff effort, test coverage, test cycle time and test efficiency. We found that with the risk-based validation approach the test efficiency has been improved by at least 50% with the same test coverage but spent only approximately 50% of the staff effort and reduced test cycle time by approximately 30%. This is because the validation scope is narrowed down, avoids repetition, and concentrates on code change areas rather than validate the whole test suite. As a result, a shorter test cycle time is required to complete the validation for the targeted milestone. Test

coverage is accumulated over different milestones and will eventually achieve 100% of the planned validation. However, the risk has been increased by 20% but this is the nature of the risk-based validation approach and it is paid-off by the ROI gained.

8 Conclusion

In this paper, we elaborated the inefficiency of traditional regression testing where time, human efforts and early defect detection are concerned, and we improve it to a more efficient way of doing testing. We aim to enhance the risk-based validation approach which is essential to optimize the test coverage with specific focus on risks analysis, task prioritization, defect prediction trend and decision making.

Risks should be properly tracked throughout the software product life cycle (SPLC). They should not be limited to software code changes but should also include all aspects that may create threats to the software validation and product release. It is no surprise that validation activity will be badly impacted when management in an organization cuts both budget and time for a software project. It is critical to use the correct skillset to turn it into success. The general methodology for this situation is not to test everything a little, but to concentrate on high-risk areas and the most defect-prone areas.

The analysis of the impact-based risk assessment is critical to the success of the project commitment and focus on minimizing the escape defects to the fields. It helps to optimize and attain complete productivity which in turn increases the results, efficacy, and efficiency of the system.

Imagine how much benefit we can gained with risk-based validation approach! The results show that it required a shorter test cycle time to complete the validation for the targeted milestone while maintaining the same test coverage and the increased risk of 20% will be contra with ROI gained. The ultimate goal in performing the risk-based validation is to achieve a project outcome that balances risks with quality, features, budget and schedule for the time, resources and efforts that have been invested in a program.

References

1 Risk Based Testing: Approach, Matrix, Process & Examples, https://www.guru99.com/risk-based-testing.html

2 Hans Schaefer, “Risk Based Software Testing: Strategies for Prioritizing Tests against Deadlines”, https://www.methodsandtools.com/archive/archive.php?id=31

3 Tricentis, “Optimize risk coverage with risk-based testing”, https://www.tricentis.com/products/automate-continuous-testing-tosca/risk-based-testing/Software Quality Days 2015, “An Exploratory Study on

4 Risk Estimation in Risk-Based Testing Approaches”, https://www.researchgate.net/publication/280948744_An_Exploratory_Study_on_Risk_Estimation_in_Risk-Based_Testing_Approaches

5 ISTQB Certified Tester Foundation Level Syllabus, 1.1. What is Testing? https://astqb.org/what-is-software-testing/

6 Dale Walter Karolak, “Software Engineering Risk Management”, IEEE Computer Society Press, 1996.

7 Ståle Amland 1999, “Risk Based Testing and Metrics”, https://sceweb.sce.uhcl.edu/helm/ROLE-Tester/myfiles/Module10/18_TST170_AppendixF.pdf

8 Derk-Jan de Grood 2010, “Advanced Testing Techniques”, https://www.istqb.org/images/Articles/deGrood_the need for selecting the right test design techniques.pdf

9 Ladislau Szilagyi 2008, “Testing Techniques and the Theory of Constraints”, https://www.istqb.org/images/Articles/szilagyi_testing Techniques and the Theory of Constraints.pdf

10 CMMI Levels of Capability and Performance, https://cmmiinstitute.com/learning/appraisals/levels

11 the Oxford English Dictionary, https://www.oxfordlearnersdictionaries.com/definition/english/the-oxford-english-dictionary
12 PMBOK® Guide and Standards, https://www.pmi.org/pmbok-guide-standards
13 Kenneth Franks 2021, “Guide to conduct a gap analysis with templates and examples”, https://www.jotform.com/gap-analysis/

A tester's appreciation of unit tests

Katie L. Fox

katiefoot9@gmail.com

Abstract

Unit tests are the foundation of the test pyramid but are commonly seen as a developer's obligation and not as a tester's secret weapon for risk-based testing. During recent interviews with Quality Engineer candidates, I heard a recurring theme that unit tests are "tests developers write," and it had me questioning my own contrary opinion. I questioned how I had arrived at those thoughts, and how I would encourage others to view them differently: as simple, powerful tests that increase in value with more visibility and more attention to detail during their creation.

Well-written unit tests provide a way to concisely understand the changes for a story without reading every line of source code. Applying risk-based principles and factoring in the areas of change indicated by unit tests could allow either for a reduction in testing scope, or it could reveal new areas that were not initially considered. It can also inform where test scenarios have already been partially or fully automated and thereby prevent duplicated effort and test suite bloat. If the "whole team owns quality," and "everyone contributes to automation," then developers and test engineers should be familiar with all levels of automated tests that ensure such quality. As part of a team's effort to shift-left, everyone needs to include focusing on creating and improving automation in areas that will gain the most benefit.

This paper will discuss the benefits of shift-left testing via tester-developer collaboration around unit tests and show its successful applications in practice within unique team structures. It will provide frameworks to help developers to write readable, atomic unit tests and testers to decide which tests to automate at each level. Additionally, this framework provides less-technical testers with conversation starters to conduct code and test inspection verbally.

Biography

Katie Fox is a senior software test engineer at e-Builder.

She spent her post-college years learning and growing within the field of software testing at Ultimate Software (now Ultimate Kronos Group) and has been navigating remote work since 2016. She graduated from the University of Central Florida with a B.S. in Computer Science and no idea that her future career existed. Fast-forward to now, she loves balancing strong domain knowledge of the user experience with a thorough understanding of the technical implementation to determine how to layer exploratory and automated testing. She takes joy in catalyzing collaborative solutions to persistent team-wide problems... if two or more people have complained about something, it is time to fix it.

When not working, she likes to run, spend days in the mountains, cook, snuggle with her cat, and cultivate plants both indoors and in the patio vegetable garden.

1 Personal career foundation

My testing career at Ultimate Software (now Ultimate Kronos Group, or UKG) began once I passed two prerequisites: an internal training on testing techniques and a technical engineering interview. The training covered functional testing techniques to design exploratory test cases without seeing the code implementation. I left the training feeling capable of writing and executing manual test cases. The technical interview covered fundamental questions on OOP and emphasized that good object-oriented design was necessary for writing testable code. After the interview, I felt responsible for knowing enough about dependency injection and inheritance to hold developers accountable for using them so that my team could write unit tests. It took a year before I put this knowledge into practice, but it instantly changed my perspective on test engineering from "testing after the code had been written" to "influencing quality within the design on the software."

In my early career, I was exposed to testing buzzwords, including *agile*, *cross-functional teams*, *test pyramid*, *software development life cycle*, and *shift left*, and learned the basics of what they meant.

- An *agile team* was responsive to changes in priority, learned details along the way instead of upfront, and broke down features into smaller pieces of work to deliver incrementally.
- A *cross-functional team* consisted of members that were business analysts, developers, and testers.
- A *test pyramid* (Fig 1) is a visual tool to represent the ideal distribution of test automation across unit, integration, and functional tests. Many unit tests form the base, fewer integration tests in the middle, and the smallest number of functional tests on top.
- *Software development life cycle* (SDLC) was the process by which a feature was delivered from the first requirements to functionality in production.
- *Shift-left* (Fig 2) meant ensuring that test automation was created early in the SDLC, so developers wrote unit tests and teams had automated tests to catch regression bugs and relieve the tester from repetitive manual test cases.

Despite understanding these techniques in isolation, it took several years of experience before I began to understand the interactions between techniques and how I could use them most effectively in my role. Once I did, I found that focusing too much on one technique can lead to gaps in quality, while focusing on using them all together can increase their effectiveness.

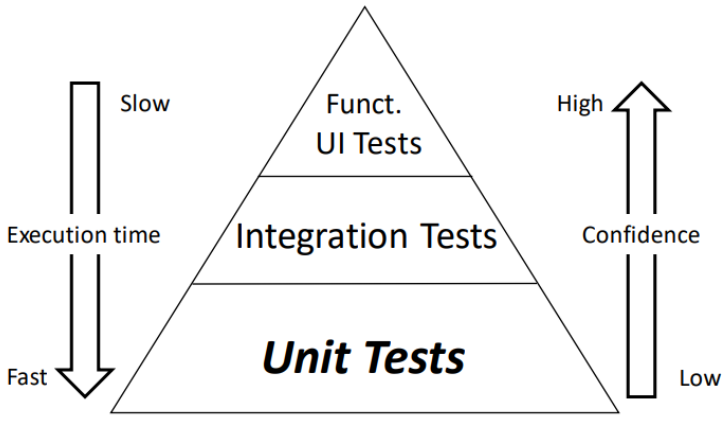


Figure 1: The Test Pyramid[1]. *Vertical arrows indicate trade-offs between fast execution in unit tests for high confidence against regression defects in functional tests.*

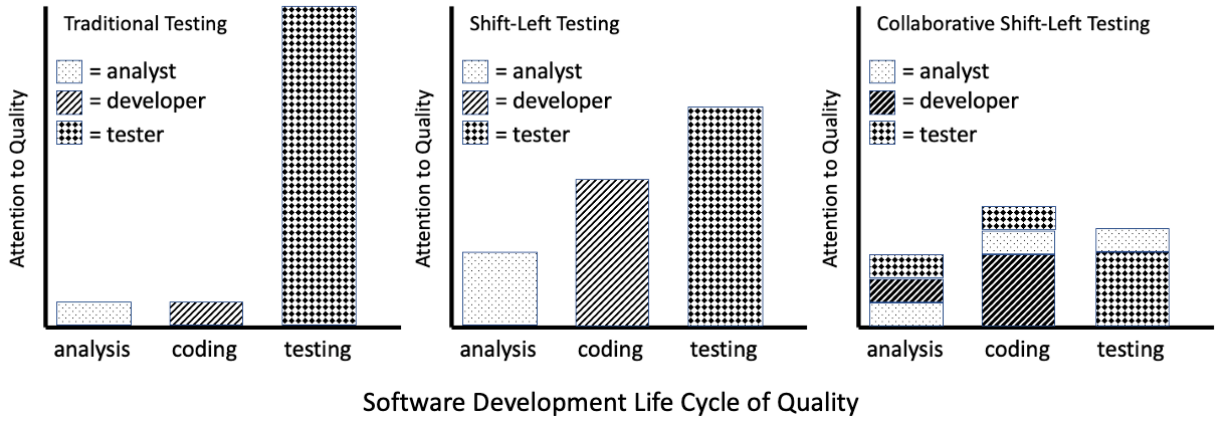


Figure 2: Attention to Quality by Role and SDLC Phase as Shift-Left Testing[2] changes are implemented. I learned shift-left as every individual role considering quality or writing tests earlier, but without the collaborative approach to testing that I found allowed us to achieve quality more efficiently.

Across the company, development teams at UKG use the Kanban agile methodology to manage their deliverables for the company's human capital management software products. This agile methodology emphasizes striving for consistency of work throughput to generate confident predictions of future delivery dates. Development teams were cross-functional and organized to work on features around particular product components. However, due to distinct/diverse technologies and testing challenges across all teams, uniformly applying agile principles and shift-left guidelines for testers was impractical, and team-specific/team-directed solutions were implemented. The flexibility afforded by team-specific testing guidelines resulted in both the challenge and freedom to figure out what worked best on an individual team.

The following procedure describes the high-level process for a development team to produce a new "feature."

1) The team business analyst would split the feature work into "stories" that represented pieces of deliverable business value.
2) Each story would undergo a mini SDLC of requirements analysis, development, and testing to ensure the delivered code meets the business need.
3) Developers would write unit tests and review code changes for each story, and then testers would conduct exploratory testing and write functional automation.
4) Once story testing was completed, the tester would perform a live demonstration of the story's business functionality to the business analyst before merging changes to the main repository.

Testers on each team managed feature test planning, functional automation strategy, and exploratory test sessions. Testing for individual stories included risk-based exploratory testing, ensuring automated coverage with new or updated tests and data, and reviewing risks with the analyst before merging the code changes. Testers also served as a source of knowledge for setting up scenarios needed to demonstrate new story functionality or reproduce defects. This consequently caused testers to frequently context-switch between roles as an individual contributor and a team-knowledge-resource. Contributing effectively as a tester, therefore, required a comprehensive understanding of their business use-cases, the underlying business rule functionality, how users interacted with the product, and how data influenced different test scenarios.

Testers cannot test everything. Automated tests provide increased confidence that code changes did not break expected behavior. In a traditional shift-left effort, developers write unit tests and testers write functional tests, but writing layers of the test pyramid in silo between roles results in a lack of cohesion within the test pyramid. Redundant scenarios or gaps in coverage occur but are difficult to identify.

The goal of this paper is to highlight the importance of collaboration between testers and developers when writing unit tests to increase the quality of team deliverables.

- Section 2 describes the structure and challenges of four of my prior teams at UKG, each with distinct team size, experience levels, and technologies. I will also describe team-level strategies for shifting left to address these challenges.
- Section 3 details specific strategies for shifting left employed across teams.
- Section 4 provides concrete examples of collaborative strategies to improve unit tests to further shift left.
- Section 5 describes ways that tester workload was reduced through collaborative unit test creation.
- Section 6 summarizes my experiences with unit tests and their potential benefits reducing the effort to achieve quality while maintaining or increasing the quality of deliverables.

2 Team Experiences

Described below are four teams at UKG Payment Services that exhibited unique structures and quality challenges which were addressed through collaborative shift-left testing, and my role in promoting this strategy to help each team deliver better quality faster.

2.1 Team 1

The team started small with four experienced developers and two novice testers with basic programming experience. We built a full-stack, event-driven web application with technology that was selected based on substantial use case research. During the months before the UI was created, testers wrote unit and integration tests for the backend services. We taught ourselves by reading existing tests for similar code, then copying and modifying them to fit the new methods. This allowed us to learn the data models and core architecture while the developers focused on writing the code and building the deployment pipeline. In the short term, a solid and coherent foundation of unit tests using consistent patterns was created and documented for future knowledge sharing. As the team doubled in size with both entry-level and established developers and testers, that foundation provided training resources in a clean code approach to aid the onboarding process. With this strong start, our primary quality challenge was to maintain cohesive testing practices as the team and codebase grew.

As part of one of the first shift-left strategies, the developers began conducting “desk checks” (post-coding demo to analyst and tester) to ensure the basic requirements satisfied the analyst’s expectation and to give the tester a head start on more complex test planning. Once the user interface development began, testers shifted unit test responsibility over to developers so that we could focus on our UI testing infrastructure. During this shift, testers collaborated with developers after the desk check to plan or review unit tests so that we knew what was covered before testing the story. Understanding where developers implemented unit tests in their code enabled testers to design more efficient integration and functional tests and ensured that each layer of tests added value.

2.2 Team 2

This second team began with an experienced group of five developers and two testers, then doubled in size within four months. The team inherited part of a full-stack, monolithic web application and was tasked with refactoring and extending its functionality to integrate with existing systems and process time-sensitive financial records. Because we did not design this solution from scratch, we spent the first months simultaneously learning its architecture, modifying code to meet business needs, and backfilling initially nonexistent test automation. Testers focused on manual exploratory testing as we familiarized ourselves with the new domain and set up UI testing infrastructure. Developers implemented necessary

features and attempted to reduce the deficit of unit and integration test coverage, but this invariably required extra effort to make the code testable. During the first year, the team focused more on keeping up with feature deliverables than improving and paying down technical debt to improve testability. The primary quality challenge was to efficiently design manual tests for each story to catch regression bugs without wasting time on irrelevant scenarios.

The first big shift-left type effort was increased collaboration in test case planning during the desk checks. This checkpoint provided an opportunity for the tester, analyst, and developer to brainstorm creative scenarios based on the story's requirements, code changes, and related process flows. After identifying what was most important to test, the developer and tester discussed whether all or part of those cases could be covered by the unit and integration tests; or, based on the code changes and existing automation coverage, if test cases could be eliminated. This process gave testers more confidence in their planning so that they could execute or automate the tests quickly and confidently without second-guessing if the coverage was sufficient.

Over time, the team caught up enough on feature deliverables to begin to shift-left in test automation. Many early unit tests had problematic design spread through copy-paste and rushed implementation. Similarly, integration tests contained repetitive hard-coded setup data that was substandard for testing code that was heavily data and state-driven. Improvements included:

1) supplying missing coverage
2) brittle tests that made refactoring difficult
3) unclear test failures that resulted in bugs when the test was deleted and re-written instead of tediously debugged
4) unreliable tests in CI that slowed down deployments

Developers and testers also regularly evaluated if the confidence gained by writing exhaustive test coverage was worth the time investment in cases where the underlying code was expected to be temporary and removed eventually. Because of the large team, frequent design improvements, and evolving test strategies, the team became challenged to keep a quality approach in sync. As individuals discovered and made improvements, they shared findings in its monthly team technical discussions so that everyone could uphold best practices in writing, maintaining, and reviewing its test suite.

2.3 Team 3

My third team started with five established developers and one novice tester. I joined them for the first four months to mentor their dedicated tester and establish shift-left testing practices so they could deliver quality quickly. We created a new microservice that compared input data from two sources and produced adjustment records to be used in a transaction reconciliation process. I started by collaborating with the developers on every story to plan or review automation. Using my time and experiences from working with the two previous teams, I established a strong foundation and corrected problematic practices early. I also emphasized whole team involvement to evaluate and improve quality practices as the codebase grew larger and more complex. By the end of my involvement, the developers worked independently on low-level automation and reviewed each other's tests, and only pulled me in when they had new questions or ideas to discuss. This allowed the other tester to continue learning the business domain and generate robust test data to expand coverage in the integration tests and exploratory manual testing.

With clean code and tests to start, the team focused our shift-left efforts on quickly catching and paying down technical debt before it could accumulate. When writing or modifying a unit test was especially complicated, we questioned if the underlying code had strayed away from SOLID[3] design principles and prioritized refactoring appropriately. If explicit data values in the test setup were not necessary for its flow, those fields were removed or assigned random values using a helper library AutoFixture[4]. For tests

asserting against List results, we leveraged a test framework FluentAssertions[5] that would produce more descriptive failure messages.

After several months, the team realized its biggest risk came from a part of our calculation code after I discovered a bug in amount rounding. The code was implemented to mimic calculation rules owned by another team because the results of those calculations in an earlier part of the process were not stored for us to simply query. When inspecting the unit tests for my team, I decided to reference unit tests from the other team to verify results in my testing and found that some of our results for a given input were different by one cent. While fixing the calculation was very simple, this bug pointed to a risk that the two calculations could easily be out of sync. After discussions with the other team, we learned a third team was also impacted because they maintained yet another copy of the rules. While the effort to centralize this code would be substantial, the risk from this technical debt discovery earned the refactoring a prioritized spot on the teams' backlogs.

2.4 Team 4

This last team was an established one of four developers that had been without a dedicated tester for a year when I joined. They formed as a DevOps subset of Team 1 and owned both the CI/CD infrastructure and a service dependency for multiple teams in the domain. This service transferred financial records from a monolithic database into our domain for additional downstream processing. It handled heavy loads during peak processing but had timing issues requiring manual intervention that we wanted to automate. While its test coverage was adequate from initial implementation, infrequent code changes resulted in those tests being overlooked as quality practices evolved over the following years. Since I had participated in development and testing during this service’s creation, I recognized many patterns that I would handle differently with my increased experience. Our quality challenge was finding ways to improve these tests while juggling necessary infrastructure changes and responding to urgent production issues.

Our first shift-left change was adding me to pull requests to review both code and test changes. At first, I familiarized myself with and evaluated test updates on my own without communicating them outward. I quickly realized I could not maintain this with new stories perpetuating the patterns I was removing. Next, I facilitated a team discussion for the developers to call out areas they had identified for inconsistent test behavior, lack of coverage, or difficult-to-maintain tests. This exercise centered quality as a full-team effort, not my agenda and sole responsibility. We met again to review easy ways to improve unit tests so the developers could use those strategies from the beginning rather than me keeping up with corrections. As my rapport on the team grew, the developers started bringing test questions proactively rather than waiting for me to add code review comments. They brought their own ideas for how to make tests better.

3 Collaborative shift-left strategies

Each of these teams had different engineering strengths and varied technical challenges. But they adopted similar strategies to shift-left in their efforts to find bugs earlier in the software development lifecycle. A dominant theme was to add checkpoints to ensure the analyst, developer, and tester were in sync and to quickly resolve any issues that surfaced since the last one. The following subsections describe the various strategies implemented and the benefits gained from each.

3.1 Story kick-offs

Teams conducted a pre-coding "kick-off" in which a business analyst would review a story's requirements with a developer and a tester. The developer identified ambiguous requirements for the analyst to clarify in real-time, which reduced the likelihood of incorrect implementation. The tester identified preliminary edge-case scenarios for the developer to account for while coding, which avoided the increased cost of finding and fixing related bugs at a later stage in the SDLC.

3.2 Testers in code reviews

Developers ensured that a tester had reviewed and approved the code review before moving a story to coding-finished status. Testers primarily reviewed unit and integration tests for consistency and adherence to test patterns that would help reduce future maintenance. Testers with less experience reading code and unit tests focused on logical test names and actionable log messages. Those with more code familiarity ensured that the conditions and assertions of the tests were appropriate for the method under test. Because tester bandwidth was limited and not every code review would receive the same level of attention, the most important outcome was that developers learn why improvements were necessary so that they could self-correct in future work.

3.3 Developer + Tester automation discussions

When automation decisions required a judgment call, such as how to handle a dependency or which test data to select, developers brought in a tester to review their options. They evaluated the pros and cons of mocking a dependency in different ways, and whether unit or integration tests were more appropriate for certain cases. They discussed expanding test coverage through more assertions, new test paths, or better test data. This collaborative approach both resolved the developer's dilemma and provided the tester with a familiarity of low-level tests that allowed them to refine exploratory testing and high-level automation scope.

3.4 Story desk-checks

Developers conducted a post-coding "desk-check" in which they would demo their working code changes to the business analyst and tester to ensure they satisfied the story requirements. They addressed edge cases that were called out during the kick-off and executed additional test scenarios that the tester identified since then. If those scenarios revealed bugs, they would be fixed before the story moved to test. Developers also used this time to alert the tester to implementation details that required a specific setup to verify in testing (e.g., processing large quantities of data in batches) so that the tester could verify and review with the analyst before completing the story.

4 Unit test improvements

My early mentors emphasized writing descriptive tests at all levels rather than documenting features in plain English. Passing tests had to reflect the current state of the code they covered, while written documentation could quickly become stale if not updated alongside every code change. Tests, therefore, should be written in a way that their failure raises questions about code changes. “Did you mean to change this behavior and forget to update the documentation (tests), or did the changes unexpectedly affect other flow?” Tests must be easily readable to aid in debugging, or they risk being ignored, deleted, or rewritten in a way that changes their intended purpose. These sections are unit test improvements I found particularly effective in speeding up my testing process as well as reducing developers' maintenance of them over time.

4.1 Name tests descriptively

Teams found it useful to name tests according to their behavior. They followed the GivenWhenThen structure to choose behavior-driven test names with information that shortened time to identify the execution path of a failed test.

Using the method UpdateRecord() as an example, consider the following test names. Initially, they seem intuitive, but they do not tell the difference between triggering an Error versus a Failure, and how those results differ. They could be mistakenly evaluated as duplicate tests and one of them incorrectly deleted.

- UpdateRecordSuccess
- UpdateRecordError
- UpdateRecordIgnore
- UpdateRecordFailure

Consider these revised names instead.

- UpdateRecord_WhenRecordFound_AndReadyToUpdate_ReturnsSuccess
- UpdateRecord_WhenRecordNotFound_ReturnsError
- UpdateRecord_WhenRecordFound_AndAlreadyUpdated_ReturnsSuccess
- UpdateRecord_WhenRecordFound_AndUpdateFails_ReturnsWarning

Now it is clear how Error and Failure are different in code path and result produced. If a test fails after refactoring, the developer immediately knows which behavior (rather than methods) was impacted and can evaluate fixes with a better understanding of the use cases that are affected.

Additionally, test names better document the behavior of the underlying method so that a new developer or tester could read through them to gain a high-level understanding rather than tracing through the method line-by-line.

4.2 Keep setup clean

Redundant or unnecessary lines of setup in the test method made it difficult to understand its purpose. While descriptive naming helped, the test body needed to be equally comprehensible, and only contain the necessities. Common issues I found were:

1) assigning unnecessary record properties or configuring dependency fakes that were not referenced in the method under test

2) assigning hard-coded values when the exact value was unnecessary for the test purpose and simply needed to be not null.

These issues typically resulted from copying code from another test fixture without removing irrelevant lines or considering if the assigned values were integral to the test flow. Over time, these small issues added up to result in tests that were very difficult to understand without running them in debug line-by-line.

A pattern that worked for my teams in our nUnit tests (since xUnit behaves differently) was to configure the system-under-test to default success conditions in the Setup methods and then use the AAA pattern (Arrange, Act, Assert) to outline each test method. Happy path tests then required minimal Arrange sections, while error paths configured only the dependency changes that triggered their specific flow. If this did not significantly reduce the Arrange section for error paths, it was a sign that the method might have too much responsibility and should be evaluated for refactoring.

Additionally, teams found value in using a library to provide random test data when the values did not drive the flow or outcome of the test. Typically, fields needed to be not-null and not-empty, and their values would be asserted in the result, but the exact values did not matter. We used AutoFixture for our .NET application, and other libraries such as Bogus or Faker exist for different languages.

Using both these strategies, to test Record.Validate(), the happy path status of "Valid" and a random value for amount are assigned in the SetUp method. Specific alternate test paths overrode status in the Arrange section to Failed or Processing before calling Record.Validate() in the Act section. The Assert section verified the result and checked that the amount field was passed through the method calls and populated in the result details, but not assert on the exact value.

4.3 Remove branching logic

Developers introduced branching logic in unit tests to maximize reuse of test code by combining similar scenarios that deviated by a single arrange or assert statement. These tests typically used a boolean parameter to trigger extra statements, which allowed two or three tests to be “simplified” down to one. This strategy reduced lines of test code in exchange for nondeterminism and future maintenance headaches, which was not desirable.

```
1.  [TestCase(true)]
2.  [TestCase(false)]
3.  public void TestErrors(bool isFatal) {
4.    //arrange
5.    if (isFatal) {
6.      //setup fatal case
7.    }
8.    ...
9.
10.   //act
11.   ...
12.
13.   //assert
14.   if (isFatal) {
15.     //logger was called
16.   }
17.   ...
18. }
```

The code snippet above shows an example where the same code is used for both a true and false test case that share identical arrange, act, assert, but then additional lines are conditional to the false case. This immediately makes the test more difficult to understand, and only worsens over time

We addressed this problem less with creativity and more with vigilance and education around the need to avoid this anti-pattern in our test suite. We used online code reviews to call out branching logic within the test and have it removed. If the developer was not yet familiar with why, a tester or another developer explained how it makes tests more difficult to read and understand and suggested appropriate ways to split the branched logic into another test.

4.4 Write better assertions

FluentAssertions[5] made a huge difference in our tests for asserting on IEnumerable objects. In the example below, I show lines 7-10 simplified to the more descriptive lines 16-17 by using the FluentAssertions library. The line 13 assertion shows a valid conversion to FluentAssertion, but one that does not take advantage of better failure messaging because it would simply produce “Expected 1 but found ___.” The FluentAssertions at the end also communicate more meaning with their structure. In this example, the importance is not that any single list element has Item1 with value 1, but that all list elements have Item1 with value 1.

```
1.  public void TestListResult() {
2.    //arrange
3.    ...
4.    //act
5.    ...
6.    //assert
7.    Assert.AreEqual(3, list.Count);
8.    Assert.AreEqual(1, list[0].Item1);
9.    Assert.AreEqual(1, list[1].Item1);
10.   Assert.AreEqual(1, list[2].Item1);
```

```
11.
12.    //this exact conversion is not an improvement
13.    list[0].Item1.Should().Be(1);
14.
15.    //simplify assertions to this
16.    list.Should().HaveCount(3);
17.    list.Should().ContainOnly(x => x.Item1 == 1);
18. }
```

The failure message for FluentAssertions will also aid in debugging. Instead of “Expected 1 but was ___” which then requires debugging to determine if one assertion or all were affected, the error would read like “Expected all elements to have Item1 but {object details} do(es) not match.” This message would contain all objects that failed the condition, immediately providing more detail on the scope of the failure to aid in identifying cause and solution.

5 Using unit tests as a tester

Easy-to-read unit tests allow for better leverage of them to logically reduce exploratory or automated testing efforts. Heavy user interaction with our front end required that we consider intuitive design and usability in addition to correct logic. This section outlines the most frequent or memorable ways that unit tests influenced testing work on a story.

5.1 Simplifying webpage testing

Many elements of webpages are enabled or disabled based on the state of records shown or selected. For example, enabling an Action button required the record to be correct status(es), and options in a dropdown list depended on the record type. This state-driven behavior was dictated by methods in the front-end code. Once we verified that all combinations for a page state result were covered in unit tests, we could apply boundary and equivalence partitioning techniques during exploratory testing to look for edge cases and usability of the front end.

5.2 State transition rules

Our backend code also had many rules governing state transition. The most common were status transitions and verifying actions could be applied, and well-written methods and tests provided peace of mind when these areas changed. Like with page state testing, unit test coverage for all combinations of input/output allowed testers to focus on looking for edge cases not accounted for by the requirements.

5.3 Known risk - batching

My teams worked on code that processed thousands of records and depended on events being raised and consumed correctly for all the records to progress as expected. To avoid excessive memory consumption, the code was designed to run queries, process records, or raise events in batches of 100 or 1000 at a time. All batches needed to run without overlap or gaps between the contents, and my teams encountered several production issues because we failed to account for new records entering the system during processing that disrupted the batch order. Batching became a known risk to make sure that developers and testers agreed on how it was both implemented and tested.

While ideally, the developer would call out this risk in one of our checkpoint meetings, unit tests gave testers a clue that this risk existed in the story so that we could initiate a testing discussion with the developer. In the FakeItEasy testing library, the method ReturnsNextFromSequence() is used to set up unique returns for multiple calls to the same dependency, and my teams used to test that a batched process occurred multiple times until it reached its typical end condition that the number of records found

was less than the batch size. Finding references to this setup provided a clue that batching was used and that the developer had considered it enough to test it as a unit, but testers would still learn more about it to determine if additional manual tests were needed.

5.4 Known risk - idempotency

Process flows triggered by an event had to be idempotent, meaning that multiple attempts to process the event should only result in one complete pass through the flow and ignore (exit out without more changes) any subsequent repetitions. Idempotent design was a pattern that testers needed to ensure but usually did not need to test edge cases except in special circumstances. Verifying the existence of unit tests for idempotent scenarios was a fast way to ensure it had been designed. I looked for tests that arranged data as if the action had already happened and then called the method again. For example, Test 3 in section 4.1 is an idempotency test I would want to find, or else ask the developer to add them as appropriate.

6 Conclusion

Shift-left means paying attention to quality earlier, finding bugs earlier, and fixing them more easily. However, simply doing quality and test steps as early as possible results in shifting fractured, siloed quality responsibility to other roles without reducing the sum effort required by the team. Taking an agile approach to shift-left, valuing “Individuals and interactions over processes and tools,”[6] results in a collaborative approach to shift-left. It allows teams to spend smaller amounts of time earlier in the software development life cycle in order to reduce both the later testing effort as well as the total quality effort across the team. Unit tests provide a collaboration point on the intersection of developer responsibility and tester specialization.

Shift-left was not accomplished by doing the exact same thing on every team. Technologies, individual skills, and preferences created different environments requiring unique approaches. On each team, I paired experience from previous teams with my growing knowledge of shift-left test principles to draw attention to the unit test foundation of our test pyramid. I strived to promote quality as a whole team effort, so process changes were discussed and decided upon by everyone. The specific changes varied by team, but on each one we streamlined our test efforts by pursuing a cohesive test pyramid, leveraging unit and integration tests to reduce the effort of manual testing and keep our functional UI tests to a minimum. The common realization on each of these teams was that, as we worked to shift-left, both testers and developers benefitted from collaborating to improve unit test strategy. The test pyramids for each team were worked on by multiple people in two different stages, and so strategies and improvements required two-way conversations. Testers helped drive that conversation, but the goal was to figure it out as a group. Unit tests provided an underrated bridge for sparking those conversations.

References

1. Fowler, Martin. 2012. “TestPyramid,” martinFowler.com, entry posted May 1, https://martinfowler.com/bliki/TestPyramid.html (Accessed August 4, 2021)

2. Smith, Larry. 2001. “Shift-Left Testing,” Dr. Dobb’s, entry posted September 1, https://www.drdobbs.com/shift-left-testing/184404768 (Accessed August 4, 2021)

3. Martin, Robert. 2009. “Getting a SOLID start,” Clean Coder, entry posted February 12, https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start (Accessed August 4, 2021)

4. FluentAssertions Tips. https://fluentassertions.com/tips/#general-tips (Accessed August 4, 2021)

5. Roberts, Jason. “.NET Unit Testing with AutoFixture”. https://www.pluralsight.com/courses/unit-testing-autofixture-dot-net (Accessed August 4, 2021)

6. Beck, K., et al. 2001. “The Agile Manifesto”. Agile Alliance. http://agilemanifesto.org/ (Accessed August 4, 2021)

Page Objects or Application Actions with Cypress, Why Not Both?

Paul Grandjean

pgrandje@gmail.com

Abstract

Cypress is the new, easy to use, end-to-end automation tool and Selenium is the hard to assemble, old grandmaster. The Cypress developers strongly suggest not to use page objects and instead use 'application actions' (custom functions) instead. With Selenium test suites, page objects are a must. But for Cypress, are application actions without page objects truly a valid recommendation for large test suites? The Cypress community presents application actions as a very useful technique for test setup, but they neglect the primary reasons for using page objects, efficient code maintenance. I believe the best practice is to use both techniques, and they are not mutually exclusive by any means. They can complement each other well. We are using page objects within the Cypress tests themselves for reducing duplicated code, managing the element locators, and writing readable tests. But for test setups, application actions as the Cypress community recommends may be the better technique by improving setup performance and eliminating dependencies on parts of the application that are also under test. This paper discusses both techniques and attempts to reconcile the discrepancy created by the Cypress developers recommending against using page objects.

Biography

Paul has been working in software QA and testing since the early 2000s and test automation since 2006. He has built both end-to-end and microservice testing frameworks and has used Selenium with multiple programming languages. In Selenium's early days Paul was a major contributor to the Selenium 1.0 documentation. He has always believed in using object-oriented programming for writing readable, well-organized tests so when he was exposed to the page object model at the first Selenium conference the technique made immediate sense. Paul has recently picked up Cypress as an alternative tool preferred by the developers on his current team. He is now enjoying bringing the strengths of both tools together for building robust end-to-end automation solutions. Earlier in Paul's career he was a high-school math teacher and then a software engineering instructor teaching object-oriented programming in C++. Passing what he's learned to others has always been a passion.

1 Introduction

Automating UI tests has come a long way since they were almost always performed manually. Selenium has been the tool of choice for well over a decade now, offering increased programmatic control from the previous generation of heavyweight, UI-based tools such as Mercury/QTP. Recent years have introduced Cypress, which arguably represents a new generation of these tools, offering a handy and light-weight test-runner with improved browser integration allowing for increased productivity with built-in debugging tools. Cypress has recently become very popular. A key Cypress advantage is UI developers already well-versed in Javascript are often open to contributing to the test suite leading to more efficiency with teams practicing “shift-left” testing.

Cypress has become the new “cool kid on the block” and appears to be increasing in acceptance. It has an easy-to-get-started GUI-interface. For those who find Selenium difficult, or have tried Selenium but didn't understand why they had 'flaky' tests, Cypress is exciting. Having used Selenium extensively but now using Cypress on my current project, I've noticed two distinct test automation cultures. And, as much as I love working with Cypress, and agree with it's effectiveness and potential increased productivity, I've seen in the Cypress community what may be a lack of understanding of what is needed to maintain large test-suites. Although Cypress has benefited and improved end-to-end testing by improving on some of Selenium's technical challenges, the promotors of Cypress appear to not understand that some of the best practices with Selenium may also apply to Cypress.

In January of this year, I joined a new team at my old company. They had not had a test automation engineer working with them previously and they had no experience in automation beyond unit tests. They had, however, begun working with Cypress and had created a few end-to-end tests of their own. I was tasked with making recommendations in how best to organize the test code, along with expanding the test coverage. This took me down a path of first, learning Cypress, and second, learning what practices from my Selenium experience would still apply. In short, I needed to study Cypress practices, compare it against my Selenium experience, and make recommendations to my team.

Then in March I changed companies, and I found myself in the same situation. I was tasked with taking a small amount of previously developed Cypress code, establish best practices, and then educate the team on the techniques we would need for a long-term and reliable test-suite.

This is the story of what I learned about what the Cypress community considers best practices, and how I had to reconcile it against my Selenium experience due to conflicting advice from the Cypress community that I felt, at least in some cases, may be wrong.

2 Tools From Different Origins, Tools From Different Communities

Selenium and Cypress have come from very different origins.

2.1 Selenium Origins

Selenium began as a JavaScript project by Jason Huggins at Thoughtworks. Improvements by additional contributors added support for multiple programming languages, which used object-oriented programming techniques in their design. Coming from these foundations is a strong understanding of object-oriented design among the Selenium community. This led to the adoption of the Page Object Model as an established best practice for writing maintainable test suites in Selenium.

Experienced Selenium test-suite developers strongly recommend using page objects in test suites, to a point where those who don't are thought of as not using well-established best practices.

2.2 Selenium Difficulties

Selenium requires some experience to use effectively. Although the techniques are well-established and code samples are easy to find, there is still a learning curve for writing reliable tests. Developers typically don't have the time to build these skills. In addition, developers are used to unit-testing tools and techniques. End-to-end testing is a very different type of testing that requires different techniques and patterns. For example, unit-testing typically uses mocks, where a small function or series of functions are tested as a small unit by a given test case. In other words, unit tests test small pieces.

In contrast, an end-to-end test case can potentially test an entire user workflow over several of the application's pages. It tests a user-initiated operation that involves the entire tech-stack from the UI all the way down to the datastore. A best practice in Selenium test suites is to use a test design pattern called the Page Object Model. This is almost always necessary for efficiently managing code changes to the tests when changes occur in the application under test. Developers (or testers) learning Selenium for the first time are sometimes not aware that to be successful this technique is necessary.

In addition, Selenium tests must use 'waiters' to properly wait for the application's DOM elements in the browser to be available to the test. This tends to be not obvious to the first time Selenium user resulting in tests that intermittently fail ('flaky tests'). It causes those new to Selenium to feel Selenium is unreliable. This is not the case, but one does need to know how to code with Selenium to ensure the tests are reliable, and this technique is not typically the first thing one learns when learning Selenium.

2.3 Enter Cypress

Cypress gives the appearance of having been developed by JavaScript UI developers who wanted to quickly write their own end-to-end tests, or UI-function tests (back-end mocked, UI fully functional). As I'm learning Cypress myself, it feels like a JavaScript tool, written by JavaScript programmers, for JavaScript programmers.

The Cypress developers clearly had some experience with Selenium when they wrote Cypress. For instance, waiting for elements in the DOM, along with asserting UI conditions such as the visibility of an element, are built into the Cypress engine. This eliminates the need to learn how to code a waiter. The result is, at least for the first time Cypress user, that their first tests are unlikely to be flaky.

Another Cypress advantage is that with Selenium the fetching of UI-elements is automatically handled with configurable timeouts. But in Cypress your *assertions are also automatically retried*. There's less coding involved, and for the experienced JavaScript programmer (which is typical of front-end developers) coding a chain of functions that both fetches the element, and asserts a condition on that element, all in one statement feels natural. You can still override the default timeout values as you can with Selenium, but the “retry-ability” (a Cypress term) is integrated into the Cypress engine to the extent that most, if not all, interactions with the application under test are automatically retried with no additional programming required. Again, another built-in feature that makes Cypress much easier for the engineer new to automated testing to build reliable tests.

In essence, the Cypress learning curve is much shorter with more immediate success and less overhead than it is with Selenium.

3 Cypress Practices Contradict Selenium Practices

3.1 Learning Cypress from a Selenium Viewpoint

One of the first things I learned when starting with Cypress though came as a surprise. The Cypress community strongly recommends NOT using the Page Object Model. Brian Mann, the founder of Cypress, says this explicitly in his 2018 conference presentation on Cypress best practices. This is an easy to find YouTube video and was one of the first Cypress resources I came across. At the beginning of his talk, he mentions “freeing yourself from page objects, or what I consider to be legacy patterns”.

I must admit this intrigued me so I watched the video. Was there something about Cypress, or JavaScript, that eliminated the need for page objects?

This was his argument. There are more efficient and direct ways of coding your tests than using page objects. In Cypress, one can take advantage of directly interacting with the JavaScript running the application, and your tests can use this to manipulate the application's state when doing any setup actions a test requires. Mr. Mann promoted what he calls “application actions” as a technique to set the state for your test, and using the application's JavaScript, to do this directly. Using this technique, one creates a custom Cypress function, rather than taking an object-oriented approach, to set the desired setup state for the test directly in the application's JavaScript code. He explains that this is much more efficient for setting up a test then using Selenium to drive the UI, for instance, filling out and submitting a form, simply to test something farther down the application's workflow.

To take this further, the Cypress website, cypress.io, directly discourages using page objects describing them as an “anti-pattern”.

> “**Anti-Pattern:** Sharing page objects, using your UI to log in, and not taking shortcuts.”

This all sounded reasonable, but my Selenium experience told me something was missing.

After viewing Brian Mann's video, I wondered why he didn't address the primary reasons test engineers use page objects:

- A single-source-of-truth for the UI element locators,
- A single-source-of-truth for coding the user-to-application interactions, and
- Test readability.

These principles are all necessary for maintaining large test suites when testing a large, and changing, application. Test engineers use page objects for eliminating code duplication which leads to effectively maintaining the test code. The video's omission of these principles caused me to wonder that although Cypress is a wonderfully useable next-generation tool, its community might be mis-advising newcomers.

I also wondered, though, if there was more to this than what could be included in a single conference presentation. Was it I who was missing something? What do other Cypress advocates teach and what are the views of Selenium users who have also used Cypress?

3.2 Not How the Selenium Community Does a Test Setup

There's even still another issue in the video that is not clear. When Mr. Mann emphasized to not use the UI, and therefore page objects, to perform test setup he implied that this is typical of how Selenium suites are constructed. But is this something that users of Selenium actually do? Well, no actually, they don't, or

at least they shouldn't. Some Selenium suites are written that way, but those who are experienced with Selenium test suites would point out it's not a good practice, for the very reasons Brian Mann states. Some organizations with Selenium suites do make this mistake when they're testing efforts rely on more junior to mid-level engineers. That is, some companies are better at implementing Selenium test suites than others. Perhaps that's the actual problem the Cypress community is referring to and that maybe the Cypress leaders have encountered that experience. But those experienced in Selenium test suites would agree with the Cypress approach. What Mr. Mann implies happens with Selenium suites is also considered a bad practice by experienced Selenium users. The issue is not that you shouldn't use page objects, it's *how* you shouldn't use page objects.

Although a Selenium suite may use different coding techniques, an experienced Selenium engineer would find a way to directly set the application's state in the test setup without using the application's UI for that setup. Why is this?

1) We shouldn't use what is being tested as part of a test setup, even for testing another part of the application. If there's a bug in the part of the application that's used for a test setup, then not only any tests testing that feature fail, but also your setup for tests of a completely different part of the application also fails. A failing setup is another thing to troubleshoot that makes your tests less reliable. You want to be troubleshooting the test failures not their setups as well. You want the tests to fail, and only fail, when they find bugs. For this reason, *setup code should be independent of the application UI being tested*.

2) Using the UI running in the browser, (and then interacting with the application's backend), is slow, REALLY SLOW! If the test just needs to set some state, do this by writing to the database directly, or writing to local storage directly, or wherever the state that the test depends on is stored. Writing the application state directly will run at the most, in a second or two, rather than often multiple seconds, even 10s of seconds, to load the browser, enter user input, click a submit button, call the backend's remote server, wait for the response, and finally update that UI within the browser.

If a Selenium suite is done correctly, its test setups will also be setting state directly. They may not do this by directly interacting with the UI's JavaScript in favor of setting database values for a true end-to-end test, but they can still set state. In addition, Selenium tests can also alter the UI's Javascript directly if they need to using the Selenium-WebDriver's executeScript() method. Essentially, a Selenium test can do the same application action style of test setup as Cypress, or it can make a call to the database (as could Cypress using a JavaScript to database client library), but in either case, the setup does not need to use page objects and shouldn't.

3.3 Should Cypress Tests Use Page Objects?

So, if the Cypress community is not fully recognizing how page objects are used, should Cypress tests be using page objects? Is the advice from the Cypress creators bad advice? Or is it just different? This question still nagged at me since I wanted to make the best recommendations to my team. And, I definitely did not want to end up refactoring a large set of tests a year or two later if I didn't find the correct recommendation. My Selenium experience told me one thing, but the Cypress community was telling me something else, and I wanted to get this right for my project.

3.3.1 What Problems Do Page Objects Actually Solve?

As mentioned earlier, we use page objects in end-to-end testing to make maintaining the test code more manageable when changes occur in the application under test. Even with small test suites, when page objects aren't used, quickly the tests end up failing and then are not promptly fixed when teams are focusing on new features, resulting in an unused, and therefore completely useless, test suite. Page

objects provide a single place to manage the interface between the tests, and the application. Then, when changes need to be made, they are made in only one place and the tests are quickly passing again.

More specifically, a page object is a class definition (or in JavaScript can simply be a module file) that supplies an interface for the tests to interact with the application being tested. All UI-element locators are encapsulated within the class (or module) and all the commands that drive the application contained by the browser (i.e. Selenium-WebDriver API calls) are contained in methods of the class. This then provides the tests an interface to the page (or components of the page) being tested.

Page Objects accomplish multiple things that make end-to-end testing much easier.

- They provide a single place for storing element locators eliminating locators copied throughout the test code. So, if the UI element in the application changes, the test code only needs to be changed in one place.
- They provide a single place for storing functional code hat simulates the user's interactions with the application. This also reduces copied code. So, if the user action changes in the application, the code simulating it only needs to change in one place.
- They add object names as “nouns” and method names as “verbs and direct-objects” to make your test code easier to read. The code will be easier to understand by someone who did not write the tests, such as a new employee or a developer if they need to fill in for the test engineer. Page objects and their methods add “semantic context”, that when done correctly, can almost allow your test cases to read like English. This is a big advantage when engineers who didn't write the original test need to maintain it, or investigate when the test fails.

There are many examples of page objects across the internet since this is such a well-established Selenium practice.

4 Researching Cypress Community Practices

Since page objects are so well-known, I assumed at least a few Cypress users must have written on this topic so I conducted some searches to collect others' opinions. I truly expected to find either a) a debate among Cypress users that had not yet settled down or b) additional JavaScript-specific techniques that replace the use of page objects. I was surprised to find neither. It seems there may be a knowledge gap between the two tools and the communities supporting them.

4.1 Cypress Blogs Also Leave Out Information

Th blog articles my searches uncovered reinforced this notion of using application actions to replace page objects. Articles such as Gleb Bahmutov's Jan 2019 post, “Stop using Page Objects and Start using App Actions” on the Cypress.io blog site. Gleb Bahmutov is another Cypress developer and appears to be a Cypress best practices promoter. The comments left by others on Mr Bahmutov's article though revealed another perspective. All the comments (about 20 at the time I read it) confirmed that although Mr Bahmutov's recommendations were correct for application actions, he was missing the primary reasons for using page objects in the first place. Here's a sample comment from “Frank F”.

> “….In my opinion, many articles written by Cypress "people" either miss the point, mix terms or seems not to know their meaning. Maybe this is because Cypress is mainly a test tool for developers, frontend developers in particular, more than a test engineering tool?”

Another article I uncovered was indicative of a more general lack of experience with Selenium causing an undeserved bias that could confuse those new test automation. An article by Paul Cowan describes how

using Selenium is subject to having “flaky”, unreliable tests and that Cypress does not have this problem. In my own practice, I've found Selenium test suites are highly reliable, *if you know how to write them*. At a certain level Mr Cowan is correct in that Cypress' retry-ability makes initial coding of tests easier, but he doesn't explain what causes Selenium tests to be flaky in the first place and that it's a problem that is easily fixed when you know how. (In Selenium waiters handling the retry-ability are implemented as part of the framework supporting the page objects.) This could be misleading to those still learning how to automate tests.

Reading through these articles, I was finding that Cypress bloggers don't address the handling of duplicated code and the code-readability that page objects provide.

4.2 Some Cypress Users Promote Page Objects but Ignore the Controversy

I did, however, find articles and videos that promote the use of page objects with Cypress. For example,

- https://www.toolsqa.com/cypress/page-object-pattern-in-cypress/
- https://testautomationu.applitools.com/cypress-tutorial/chapter7.html

This was encouraging, however these authors did not appear to be involved directly with the Cypress project itself. So this does not negate the fact that the Cypress founders themselves might be promoting patterns that could lead to bad practices. These types of articles, and videos, demonstrated how to implement page objects in Cypress, but they do not even mention that this contradicts what the Cypress developers recommend. Basically, they teach how to create a page object, without question *should you create a page object* in Cypress? So once again, there appears to be a disconnect from this group of Cypress users as well. It was as if they had learned Cypress from the cypress.io documentation, without having viewed the “best practices” videos or blog articles.

4.3 Some Present Cypress vs Selenium Differences, but Ignore the Controversy

Finally, another type of article or video I found, was those that present the differences, and tradeoffs between Selenium and Cypress. These ignored page objects entirely. They focussed on the technical differences, and on each tools strengths and weaknesses along with how to choose between the tools in different situations. No where did they mention page objects, application actions, or any trade-offs between varying best practices. Once again, I found the best practices discrepancies were not addressed or even recognized.

5 Resolving The Controversy

My research told me no one is writing about these conflicting practices and trying to reconcile them. As a former teacher, and as an engineer who sometimes mentors others, this concerns me. It has the potential to mislead those who are new to automation. It potentially could lead to test suites that quickly become difficult to maintain, out of date, and eventually unusable. Or, possibly, if the Cypress developers are correct, there could be many Cypress test suites using page objects unnecessarily, creating unnecessary layers of code leading to inefficiencies in test development and execution.

5.1 What We're Doing on My Team

After about a week of research (along with similar but less thorough research at my previous company) I was able to make recommendations to my team. I recommended that we should in fact use page objects with our Cypress tests. I didn't find any clear reasons not to, and it appeared all the reasons to do so still applied.

I also recommended that when we need to set state in a test setup that we use application actions. At the time of writing, however, my team has not needed any special setup steps. Our team's responsibility are for a “read-only” portion of the application which doesn't require setting state for our test cases. The data we depend on for is provided as mocks which we control in our test suite. (Other teams at Driveway.com will have this situation. One team is using application actions and the other teams have not yet started they're test suites).

We do however, test pages where we expect changes will occur to the UI. This determined our decision to use page objects so we can control page changes in one place. My front-end develops who contribute to our Cypress tests we're enthusiastic when I presented page objects to them and we have incorporated the pattern in our tests as a team. We now have a few page objects implemented and most of our legacy tests have been refactored to use them. The front-end developers I work with have told me they like the technique and have been supportive and at times contributed to the page objects. For me, I now have test cases I can more easily read even when they were written by one of the developers.

5.2 What We're Doing at Driveway.com

Although my team has adopted page objects, this is by no means the practice throughout the Driveway.com's other teams.

We have multiple teams developing different aspects of Driveway.com along with the internal administrative applications that support the website and it's data. Our other teams have not adopted the page object model. A couple of teams are this time following the advice of the Cypress developers. Others have not yet started their Cypress automation. And, finally, we have a team that has specifically decided not to use page objects and at one point recommended we don't as well, feeling they are not necessary and that they add complexity with little gain. As an organization we have accepted that there will be differences in “best practices” among our times. It will take time for us to learn what the actual best practices for Driveway.com will be. Or, if those best practices truly need to be different from team to team.

5.3 Are Page Objects Truly Necessary in Cypress?

I still feel page objects are highly useful if not actually necessary. But as our test suite at Driveway.com grows, and as we gain more experience using our suite, this question will have to be answered for us over time. Although page objects are usually necessary in a Selenium suite, it's possible it may not be quite as necessary with Cypress. Cypress does tend to make test coding easier than in a Selenium suite, at least it does for those already comfortable with Javascript.

I do feel, however, that even if no longer necessary, the page objects add much readability to our test cases. They do this by providing semantic context through using the names of the pages, and components on the page within the test case along with using action names for the user interactions with that page. As I mentioned earlier, my team likes the technique. We will simply have to see over time if our other Driveway teams encounter problems in there tests that cause them to eventually adopt page objects, Or, if the keeping their tests updated ends up not being a significant problem, perhaps I become convinced page objects are still useful, but not necessary as they are with Selenium. I'm looking forward to seeing how we feel as a group at Driveway a year from now.

6 Conclusions

I won't yet say the Cypress developers are wrong. As Cypress is a new tool to me, and JavaScript is a newer language to me, there's always a chance that I could be missing something. However, with the research I've done up to the present, I cannot recommend to anyone using Cypress to not use page objects. As with Selenium, I would recommend they use page objects to manage the locators and user

interactions, provide semantics for readability, but to do this for driving the test itself and not for the test setup. I would also recommend the use of application actions for making the test setups efficient. Essentially, I would recommend both techniques and not to ignore either one, but to use them appropriately in the places where they are intended to be used.

At this time, my research has caused me to conclude

- The Cypress community does not address the actual reasons why we use page objects.
- No one is directly addressing this omission by the Cypress community.
- No one in either community is addressing the differences in practices.
- The Cypress community is correct to promote application actions for test setup and the Selenium community could benefit by making this more explicit.

Which test design pattern is right? Both! But like any tool, use page objects and application actions in the places where they are designed to be most effective.

References

Alam, Bushra (2020, May 27). Cypress - Page Object Model | Part 15. You Tube. https://www.youtube.com/watch?v=5ODb5eQTB44.

Bahlaik, Kushal (2020, Oct 4). Building a Test Automation Framework using Cypress.io — Adding Page Object Model (POM) (Part 4). Medium.com. https://medium.com/omnius/building-a-test-automation-framework-using-cypress-io-adding-page-object-model-pom-part-4-1e6742456515.

Bahmutov, Gleb (2019, Jan 3). Stop Using Page Objects and Start Using App Applications. Cypress.io Blogs. https://www.cypress.io/blog/2019/01/03/stop-using-page-objects-and-start-using-app-actions/.

Bhimavarapu, Pavan Kumar (2019, Dec 27). Par 18: Page Object Mode (POM) in Cypress. YouTube. https://www.youtube.com/watch?v=5ifXs65O36k.

Cowan, Paul (2021, Jan 1). Cypress vs. Selenium: Why Cypress is the Better Option. LogRocket. https://blog.logrocket.com/cypress-io-the-selenium-killer/.

Cypress.io. https://cypress.io.

Mann, Brian (2018, Mar 5). Cypress Best Practices, Assert(JS) Conference. YouTube. https://www.youtube.com/watch?v=5XQOK0v_YRE.

Nunes, Diego (2020, Mar 2). Cypress: Page Objects vs App Actions. The Geeky Gecko. https://www.diogonunes.com/blog/cypress-pageobjects-vs-appactions/.

Sheth, Himansu (2020, Nov 12). Selenium vs Cypress – Which Is Better in 2021? Lambda Test. https://www.lambdatest.com/blog/cypress-vs-selenium-comparison/.

Testim (author unknown) (2021, June 11). What Is Shift Left Testing? A Guide to Improving Your QA. June 11, 2021, https://www.testim.io/blog/shift-left-testing-guide/.

Selenium.dev. Selenium History. https://www.selenium.dev/history/.

Yaseen, Mohamed (2021, April 19). Cypress vs Selenium. Medium.com. https://medium.com/nerd-for-tech/cypress-vs-selenium-32677b09ea5b.

Crafting A Quality Organization Through Proof of Concepts

Paul Grimes

p.grimes@pokemon.com

Abstract

Organizations often make changes without a way to evaluate if they are getting the results they want. This can be true when incorporating a new tool, making changes to communication routines, or changing management structure.
The Product quality and Insights team at the Pokémon company international uses Proof of Concepts (POC's) direct changes we want to make. Anyone on the team can write up a POC proposal for an idea they believe will make the team better. The proposal includes what they want to change, how long the POC will take, and what questions the submitter hopes to answer. The POC is reviewed and refined to determine the criteria for success. If accepted the POC is then executed and evaluated and if successful, the change is adopted. This paper will look at both successful and unsuccessful POC's, what was learned from each, and how using the POC process gives the team ownership of day-to-day work and the structure of the PQI team.

Biography

Paul Grimes has worked in the software industry over twenty years. He loves technology and has worked in many vanguard fields including web services, game network engineering, automation planning and automation implementation. He spent the early part of his career at Microsoft working on Office, then shifted to work on PC and Xbox games. He also has varied experience from working at Barnes and Noble, Disney and various startups in the games and data knowledge industries. He currently works for The Pokémon Company International as a Senior Software Development Engineer in Test.

1 Introduction

The Pokémon Company International (TPCi) is a subsidiary of The Pokémon Company in Japan. It is responsible for brand management, licensing and marketing, the Pokémon Trading Card Game, the animated TV series, home entertainment, and the official Pokémon website.

To support the brand, TPCi has a dedicated technology organization that is responsible for many internal tools and external offerings.

Pokemon.com is the official website for the Pokémon brand and is available around the world. The site features dedicated pages that host information about certain products, such as the Pokémon video games and the Pokémon Trading Card Game (TCG). In addition, Pokémon TV is where fans can watch their favorite episodes of the animated TV series or select animated movies, and the Pokédex can help fans find information about their favorite Pokémon.

Pokémon Trainer Club (PTC) allows fans to manage Pokemon.com profiles for adults and children. PTC also allows players to track their progress in tournaments for both the video game and the Pokémon TCG, sign into Pokémon apps like Pokémon GO, play the Pokémon TCG Online, or track which episode of the Pokémon animated series they are watching on Pokémon TV.

2 Pokémon's Product Quality and Insights Team

Our Product Quality and Insights (PQI) team operates inside the technology organization and focuses on ensuring great experiences for our customers. The team is comprised of our customer service team that engages with our customers to solve problems and address issues and our quality team that focuses on improving development and product quality practices as products are developed. The quality team focuses on working with internal teams on the development of projects, improving the development process, and validating releases by ensuring we are applying the right validation at the right times.

2.1 PQI Team Values

The PQI team is organized around six core values developed by the PQI and our leadership group. They help us inform our decisions. They have been amended and added to by the team as we have changed and grown. We recognize team members who demonstrate these values at monthly team meetings. Our team values are:

Great Triumph: We believe every member of our team is working to deliver amazing, memorable experiences. Valuing everyone's opinions and viewpoints will help us reach our goal.

Customer First: Using customer-centric practices in our work, we are committed to providing a quality product and positive experience to our customers and fans.

Continued Learning: We strive to better ourselves every day. We will do this by continuing to learn new things, take on new tasks, and by asking questions. We're ok with trying something different and failing. I'll know what to do better the next time.

People Matter: Our team is a family. It is super important to us to care for each other, even outside of work. We believe the people make the company great and treat them accordingly.

Transparent Communication: We are honest and transparent. When discussing the quality of a product, we will use data to communicate the quality. Obfuscating the true quality of a product isn't productive.

Courageous Action: We act without fear of impediments or dissenting opinions. We confront risks with prudence and tenacity even if they leave us vulnerable. We support each other because we trust that our actions improve quality for everyone.

3 Proof Of Concept (POC)

Proof of concepts have been part of engineering since the 1960's (Merriam-Webster.com Dictionary n.d.). A POC is a small, sometimes incomplete real-world development of an idea to a working state so it can be evaluated. These experiments are intended to test both the feasibility of an idea and expose challenges in the implementation of the idea. They take a hypothetical and put it into practice so it can be interacted with.

POC's differ from prototypes, although the terms are sometimes used interchangeably (The Motley Fool 2021). A POC demonstrates that something can be done, while a prototype shows how it will be developed. A POC should demonstrate that a solution can be achieved, while a prototype shows how a product will operate. Because of this difference, a POC is often smaller in scale and has a smaller cost than a prototype.

4 The POC Process at TPCi

We create POC's for many different ideas in the PQI team, such as acquiring new software, developing a tool, or changing a team process such as our team standup meetings. We choose to do POC's instead of prototypes because they are smaller in scope and more manageable. POC's are easy to start as an individual and they are easy to pitch to the team while still having a measurable effect.

POC's help the PQI team live our values. They enable great triumph by allowing us to have ownership over our organization and the tools we use. POC's allow everyone have an equal voice and propose change when they find a challenge that a team member believes they have a solution for. POC's let us communicate about what is working for our team both technically and culturally. Our last team value, courageous action, was added as the result of a POC.

4.1 A Template for POC Proposals

We have defined a template for POC's so that we can evaluate them fairly on consistent terms and determine which ones to move forward with and with what priority.

PROCESS/PROJECT	NAME OF CONCEPT
DESCRIPTION	A lightweight description of the concept and why it is something to try out and experiment with.
PURPOSE	What do you hope to get out of this PoC? Why do you want to run it?
EXPECTED LENGTH	Duration of the PoC until a final assessment is done.
HYPOTHESIS & MEASURES	1. **Hypothesis**: Prediction of what will change as a result of this PoC

	1. **Measure**: How you plan on measuring or validating the above hypothesis 2. **Hypothesis**: Another Hypothesis (if any) 2. **Measure**: Another measure (if any) N. **Hypothesis**: Continue adding as many hypotheses and measurements as the PoC requires based on its purpose. N. **Measure**: Continue adding as many hypotheses and measurements as the PoC requires based on its purpose.
RESULTS/REFLECTIONS	Final conclusions after the end of the PoC. Did it meet expectations? Prove or disprove the hypothesis? What did you learn and what are the takeaways or next steps?

Figure 1. A Copy of the PQI POC template

The template allows the author to describe the proposal, define the length, propose some hypotheses on the outcomes of the POC as well the measurements that will evaluate the outcomes. By having this in the template it encourages authors to think about why the POC is valuable and how its success will be measured. After the POC, a results and reflections are documented so the process is tracked, and any adjustments to the original plan can be noted.

The same process and template is used for both technical and non-technical POCs. The goal is to try an idea that will improve the group in a meaningful way. This can occur including acquiring a new tool or technology, changing how we communicate, how we are organized, or experimenting with how we work with partners.

4.2 The POC Process

A POC proposal is made by a PQI team member by filling out the template and posting it on the wiki. While the form is short it encourages the team member submitting the proposal to consider their idea more fully. They must understand the purpose of the idea they are proposing and consider its challenges. The team member must determine what they believe the outcomes will be, and how they are going to objectively measure those outcomes.

When writing a POC it can be difficult to determine how long a POC should go on. This varies POC to POC. For technical POC's where you are trying a product, it may be dictated by the trial length of the tool or technology. For cultural things, it often makes sense for it to go on through several "useful cycles". For example, if something is supposed to occur monthly, the POC may call for a timeline of 3 months so that you attempt 3 iterations before trying to draw conclusions. Timelines should be as short as possible while giving a fair evaluation of the concept.

The POC proposal is then reviewed by the management team as well as a subset of the team for consideration. POC's can be iterated upon in the proposal stage for clarity of purpose and objectives. Once approved the original requester will move forward with implementation, working with the leadership team to acquire the resources needed.

After a POC is complete, the creator returns to the template and fills out the RESULTS/REFLECTIONS section. They recall how the POC process went and develop how to proceed. If the POC was successful, a plan is developed to adopt the practice or tool. It the POC was not successful, they might revamp the POC and try again or it may be archived so the team is aware of what has been tried before. This often saves the team time, so they do not spend time revaluating ideas that did not meet the team's needs. It also provides inspiration for different approaches to take considering previous data. All POC's are archived on the internal wiki for future reference regardless of their outcome.

5 POC Examples

5.1 Technical Example: mabl

mabl is a test automation product that helps teams create, edit and execute UI automation. The PQI team decided to execute a POC to determine if mabl would meet the needs for UI automation (mabl Inc. n.d.).

PROCESS/PROJECT	Automated testing through AI assisted testing
DESCRIPTION	Pilot using mabl (http://mabl.com) for Pokémon testing by using the PTV HTML5 site
PURPOSE	We should evaluate if there are tools that can help us test smarter and allow us to cover regression with fewer people devoted to it. We could use a small, contained project with known constraints to see if we can benefit from AI assisted tools.
EXPECTED LENGTH	1 month
HYPOTHESIS & MEASURES	1. **Hypothesis**: That a tool like mabl could reduce the number of tests cases we run manually by giving adequate coverage on regression 1. **Measure**: Perform a test pass without mabl and record the number of cases run by hand. Use mabl to automate a set of journeys. Update the site content. Run a test pass by hand, note which ones were covered by mabl and which ones were not. Note if mabl missed any errors caught by the test pass. Calculate the percentage of tests that we have confidence that mabl could perform on its own.

Figure 2. The POC template for the mabl POC

The drive behind this POC was to do a technical evaluation of mabl. We determined a small web-based project at TPCi that could be used to exercise the mabl platform. When setting off to execute the POC we worked with the team at mabl to address questions about the technology and do small working sessions so that TPCI employees came up to speed quickly and learned best practices from the mabl team. We also brought in experts in our current web automation solution to give an analysis of the toolset and compare what their process was to what it would be if we started supplementing their work with mabl.

The team created a suite of tests for the Pokémon TV web experience. Tests were created for both English and localized versions of the site. The site often updates content, so the team was able to evaluate how mabl handles changing content, one of mabl's key selling points. The team focused on the PQI teams core scenarios that they felt they would use mabl for, regression scenarios on a site that updates frequently. A comparison between a mabl run and a run done was performed and the automation was found to significantly reduce the execution effort,

In executing the POC, the PQI team discovered that mabl met the needs of the team for implementing, executing and maintaining web UI automation. The POC timeline was extended as the POC was being performed to address some technical issues and because early results showed promise. When the metrics set out in the POC were evaluated, the team found that the number of tests that could be authored and maintained using mabl far exceeded what the team would have been able to develop without the tool in the same amount of time. They also required less maintenance than tests developed using other frameworks. The team was able to create and execute automation against both development and production environments as required and were able to compare our manual testing process versus the automated approach to validation of changes.

5.2 Cultural Example: Agile Strike Teams

POCs are used for technical changes or adopting new technologies by many teams cross the industry. One unique aspect of the way the PQI team have implemented POC's at TPCI is that they can be used for cultural and organizational changes as well.

Our team framework needed more flexibility to be able to meet partner team needs. After considering multiple options including reorganization, the PQI team decided to do a POC for strike teams. These are scrum teams focused on a single area of a product such as writing regression tests. The scrum team brings together all the skills needed to perform a function.

PROCESS/PROJECT	Agile Strike Team
DESCRIPTION	Build out a team based on the AGILE process of development for a self contained project
PURPOSE	After Scrum master training and a deeper exploration of the AGILE process, the purpose of this project is to actually implement an AGILE team at TPCi and see how effective it truly is. This POC also has a related purpose of exploring the effectiveness of an internal AGILE team as a "strike" team, that can rapidly ramp up and deliver value in high need areas in a relatively short period of time.
EXPECTED LENGTH	4 months
HYPOTHESIS & MEASURES	1. **Hypothesis**: Leveraging AGILE methodologies will enable a team to deliver concrete results on a broad/nebulous project incrementally. 1. **Measure**: All or the vast majority of sprint reviews have something demonstrable to show stakeholders that can be consumed by them. 2. **Hypothesis:** Following the AGILE methodology enables a small team to deliver significantly more work more quickly for a self contained project than other strategies.

	2. **Measure:** Estimate how long we think it will take to do the project ahead of time, and then check in on both progress and re-estimate project completion time at the end of the POC. 3. **Hypothesis:** Team collaboration, growth, communication and support will improve between the members of the AGILE team and will persist after the completion of the project. 3. **Measure:** The other measures will help measure this, in addition we will check in on the team's overall feeling of inter-team support and collaboration both before the start of this project and at the end of the POC.

Figure 3. The POC template for Agile Strike Teams

In implementation a full scrum team was assembled. This included a dedicated Product Owner, Scrum Master and a small number of team members. This team kept its own backlog, determined its own sprint schedule, and ran a full set of ceremonies every sprint (sprint planning, backlog refinement, and retrospective). The team picked a self contained project of writing a number or automated regression cases for the pokemon.com website. This self-contained project allowed the team to evaluate the agile practices versus the more traditional practices that had been done prior on the project.

This team was able to create a large amount of regression automation for our Pokémon.com website, greatly reducing the burden of manual tests for update to the site. We also discovered that our agile team was able to fully implement scrum, even while our partner teams were not following scrum. Because of this POC we now have more teams operating inside the scrum framework, and partner teams integrating with those scrum teams effectively.

When evaluating how the project performed versus the measurements set out in the POC, the results were very good. Every sprint review had demonstrable results presented. Because this was done in an agile fashion using 2 week sprints, the team was able to incorporate feedback from those reviews to keep focused on delivering the most impactful work each sprint. The team, while unable to meet their original estimate of 4 months, was able to predictably deliver a set amount of work each sprint. For the communication and collaboration metrics, it was found the team internally far preferred working in a strict SCRUM framework and felt much more collaborative working this way. When discussing the team with partners, the team was regarded as very reliable and predictable, delivering excellent results.

5.3 Cultural Example: Team Announcements

Even small changes to team dynamics can be tried and evaluated using a POC.

One of the struggles of our team during the challenges of 2020-2021 has been keeping connected. Our team size has grown during this time frame from ~20 members to 40 members of PQI. Team members were working remotely full time, which was a first for TPCi. Team members were being interviewed and on-boarded remotely. To make sure people were aware of happenings inside the team, one of our members suggested creating a Slack channel specifically for team announcements.

PROCESS/PROJECT	**#tech-test-cs-announcements**

DESCRIPTION	An open, non-private, Slack channel for the Test and CS team to broadcast announcements that impact team members and stakeholders.
PURPOSE	A centralized location for announcements, so that reminders, out of office mentions and, other broadcasts are easily viewable and are not lost. Types of Announcements that I see fit in this channel. **Product Launch Announcements:** • Marketing content. • PTV Episodes or new content. • PTCS/PCOM deployments • BOBs functionality. **Event Attendance:** • StarWest, pycon, etc • Training sessions; Agile Workshop, Chef training, etc • LinkedIn Learning dedicated time for career growth **New Hires:** • I realize that [our director] puts this in email form, this may be another landing pad for this content. **Appreciations:** • During WFH our Test Shards have made this a little more difficult to surface to the larger team, so this may be an area where an appreciation can be given. With the caveat that threading is disabled so information is front and center in a "read only" form. I think emojis are able to be posted. **Work Anniversary:** • Celebratory reminders for individuals on the Test and CS team **Holiday Reminders:** • Setup Slack reminders the week of or before hand that the date is approaching.

	Out of Office Reminders: • Use of reminder syntax, or Slack has made it easier with a date picker to select when the reminder should be on.
EXPECTED LENGTH	2 months.
HYPOTHESIS & MEASURES	1. **Hypothesis**: Before the team begins to use the channel, setting up prime examples of how the channel should be used will set it up for success as my hope is that "announcements" are formal. I do not for see this channel as the place to say "Good morning, Lunch, Away, Back, Good Evening." I still see tech-test-cs as a common ground channel for lively discussions. The announcements channel which is an open channel to join and can be shared out to Partners or Stakeholders as a **Read Only** channel for them, is a place where any one can quickly acquire information on the Test and CS team. 1. **Measure**: SurveyMonkey questionnaire.

Figure 4. The POC template for Announcements Slack channel

When the team evaluated the POC, the PQI managers had concern about adding another Slack channel. We already had a channel for PQI wide communication. The POC was altered to do recurring announcements using Slack automation in the existing channel. The team decided to alter the POC and reduce its scope to evaluate its effectiveness.

Company holidays were added as reminders from Slack in the existing channel. Personal milestones, like work anniversary dates and birthdays were added for each member of the PQI team. These come as announcements from the automation highlighting team member milestones.

Slackbot 9:00 AM
Reminder: **@here** “Today is [@emloyee’s] work-anniversary, please join me in congratulating him for his service and contributions since 8/14/2017” eve.

Figure 5. A sample milestone announcement

Ultimately some of the ideas of the POC were implemented, but some aspects, such as Product information such as launches or other milestones were not. In this example, the team should revisit the stated purpose of the POC and see if there are ways to accomplish those, either in another POC, or by further revising this one.

For the reduced scope, this POC has been effective. The announcements in the channel draw reactions from the team. The announcements are performing their role in helping the team recognize and support one and other.

6 Conclusions

By using POC's, the PQI team has been able to remain flexible and meet the changing demands of TPCi while maintaining ownership over our processes, tools, and culture. Taking the evaluation process usually reserved for technical evaluations and using them for cultural and process changes allows us to evaluate the changes fairly and adjust their course. By requiring most changes to start with a POC proposal in a standardized template, we ensure that we know what the goals of the changes we are making are, and how to evaluate them. The process of reviewing and approving the POC's allows the team to prioritize which projects to attempt with the available time and resources. By applying rigor to our decisions of how and when to change, the group gets more investment in changes from team members because the changes have been vetted by the team.

While well-known and used for technical experiments and initiatives, the PQI teams use of POCs for evaluation of organizational and cultural changes appears relatively unique. Since 2018 the PQI team has had more than 40 POC proposals. More than half of the proposals have been for cultural, or process changes rather than technical implementations or tool evaluations. This demonstrates the teams desire to improve how we work and the environment we work in. The team knows that people matter and how we work is as important to our long-term employee satisfaction as what we accomplish. The ability for every member of the team to propose change and for the changes to be evaluated in a quantitative framework allows every member of the team to take courageous action to make changes and shape the organization we work into better suit us as the work and work environment change.

References

mabl Inc. n.d. *Functional test automation for creating reliable end-to-end tests.* Accessed August 30, 2021. https://www.mabl.com/product.

Merriam-Webster.com Dictionary. n.d. *proof of concept.* Accessed July 28, 2021. https://www.merriam-webster.com/dictionary/proof%20of%20concept.

The Motley Fool. 2021. *Your Complete Guide to Proof of Concept.* January 2. Accessed July 28, 2021. https://www.fool.com/the-blueprint/proof-of-concept/.

Security in IPv6 enabled home networks: Are we ready yet?

Nikhitha Kishore

Nikhitha_kishore@mcafee.com

Abstract

IPv6 was developed by the Internet Engineering Task Force (IETF) as a solution to IPv4 address exhaustion problem. The exponential increase in the number of internet devices amplified the need to transition to networks that have addresses larger than the IPv4's 32-bit address space. IPv6 provides a 128-bit address space which is sufficient to provide addresses for any conceivable number of individuals, organizations, devices, or network enabled objects in the foreseeable future.

ISPs have recognized the importance of IPv6 and most of the major Internet Players have already started the transition. Since it is a longer journey towards an IPv6 only world, most of the devices and networks are made to support dual stack (IPv4 + IPv6). This means that a lot of home networks are in the IPv6 network space by now. IPv6 significantly increases the complexity of home networks. And complexity is the enemy of security. Quite often connecting home networks with IPv6 takes precedence over the security changes from IPv4 to IPv6. Functionality is important but should not come at the expense of home network security. Transition issues and challenges in deployment can cause security issues if not addressed thoroughly. Also, the scarcity of IPv6 tools to monitor/report the network security can lead to delayed responses in case of a network attack.

Addressing IPv6 security issues is imperative in home-networks especially due to the increasing usage of IOT devices. Since installing endpoint security might not be possible in a lot of IOT devices the user is at the mercy of the network security handling offered by the ISP. It is unfair to expect most of the users to have knowledge about these attacks.

This paper details the security challenges introduced with IPv6 and some of the attacks possible in an IPv6 home network. It also details some of the existing tools and solutions that can be helpful to detect the security issues in such networks.

Biography

Nikhitha Kishore is working as a Lead SDET in McAfee India for the past six years and has worked with multiple global ISPs in integrating McAfee's home security solution with their routers. Most of her work revolves around testing network security solutions based on DNS, DoT/DoH, SNI, HTTP inspection etc. Prior to McAfee, Nikhitha has worked in Novell Software, India. Network Security has always been her core interest and while doing her masters in NIT Bhopal she has published two papers in international journals (IET, IEEE explore) on the topic of delay tolerant networks.

1 Introduction

IP addresses are assigned to every connected device in the internet for host identification and location addressing. Before the commercialization of the Internet which began in the 1990's, the address space offered by the 32-bit IPv4 was sufficient to assign IP addresses for the interconnection of regional academia and military networks. The transition to the commercial Internet saw an exponential growth in the number of devices connected to the Internet starting from personal and mobile devices to the current boom of the smart IOT devices (Figure 1). The ubiquity of Internet soon led to the depletion of the 4.3 billion unique addresses supported by IPv4 and a new version of IP (IPv6) was standardized in 1998.

Though the official IPv6 deployment started in June 2012, recent years have seen the adoption of IPv6 gaining traction due to the widespread usage of networking devices in the consumer space. Some of the existing consumer devices like refrigerators, microwave ovens and air conditioners have transitioned to the smart home market. The omnipresence of smart home devices has led to many houses using home automation setups which helps the homeowners to control their internet-connected devices remotely. Smart home automation devices connect appliances, switches, and gadgets to a central hub, enabling them to control these devices conveniently even if the homeowner is away. A smart home setup can command groups of lights to turn on or off when the front door opens, receive alerts when a leak springs, or program the thermostat to work with the AC when the room gets hot. Without a doubt, the consumer devices connected to the Internet are exponentially growing. ISPs have addressed this growth by rapidly transitioning to IPv6 networks in the last few years.

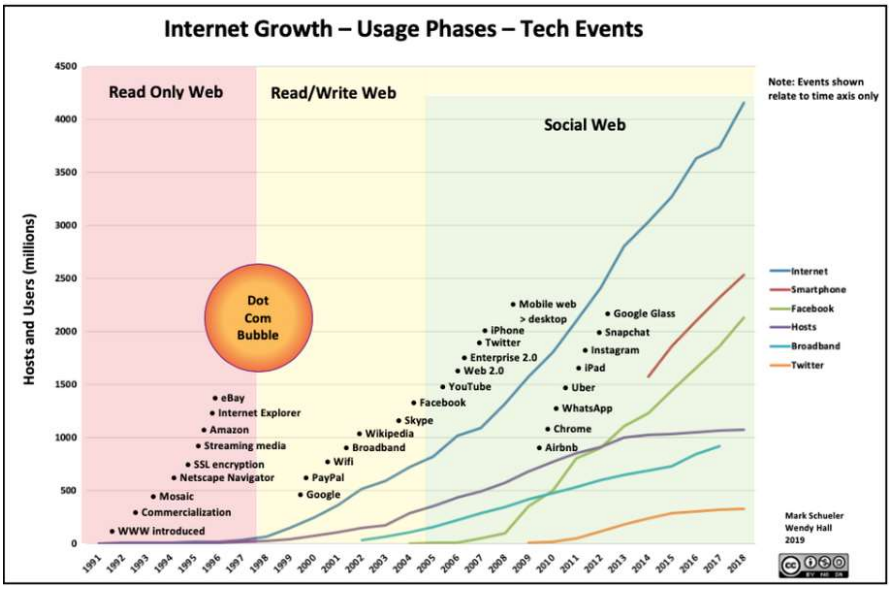


Figure 1: Growth of Internet since 1991

When home networks grow with the addition of numerous smart devices, the attack surface also increases. A typical smart home network consists of PCs and printers connected over wired networks, wireless devices like laptops, gaming consoles and wearable gadgets, and IOT devices like smart doorbell, thermostat, voice controllers, smart light switches etc. (Figure 2). Many of these devices automatically connect to the Internet for updates, statistics, and diagnostics collection. Users unaware of the ISP's IPv6 deployment introduces more attack vectors to their home network.

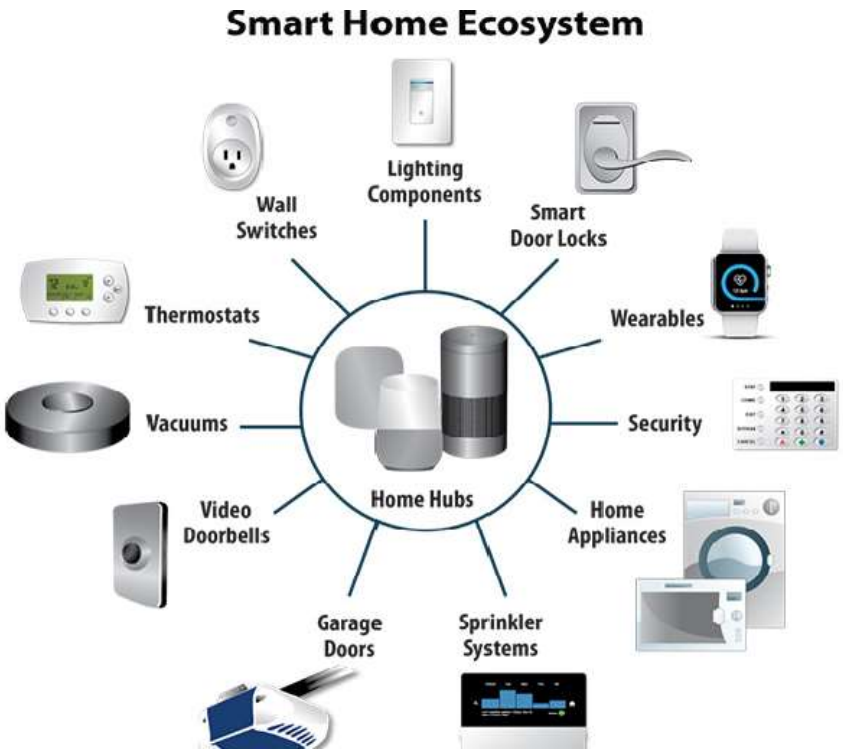


Figure 2: Smart Home Ecosystem

The dawn of IPv6 era, comes with its own set of security risks/challenges. It is a matter of time by which the attacks in IPv4 transforms to the IPv6 networks. The concern comes out of the fact that there are plenty of solutions, prevention mechanisms and detection tools available for IPv4 owing to its long reign, but the same might not be true for IPv6. Imagine a user who has been relying on a security solution in his home which is only IPv4 supported. What happens to such home networks when ISP deploys IPv6? Most of the users might be unaware of what an IPv6 deployment means to their networks and might not be educated enough by their ISPs to upgrade their security solutions to support dual stack networks. Along with this, the lack of trained professionals and tools to monitor IPv6 networks will add significant delay in the time a security issue is detected and reported.

In conclusion, though IPv6 is a necessity, not addressing the security concerns can severely impact its success. Attackers are going to adapt the attacks in IPv4 to IPv6 and it is important to dedicate sufficient resources in preventing these attacks. IPv6 support and testing the security solutions for home networks requires more traction to help the world transition to a haggle free IPv6 era.

2 IPv6 Overview

Let’s understand the basics of IPv6 and its significance.

2.1 What is IPv6?

IPv6 was developed because IPv4 does not have enough addresses available to sustain the ever-growing internet and all the devices that need a unique IP address to connect to it. IPv4 has a theoretical upper limit of about 4 billion (4,000,000,000) unique addresses but in practice IPv4 is unlikely to support a sustainable population of no more than about 250 million uniquely addressed nodes.

In contrast to IPv4 addresses, which use only 32 bits, IPv6 addresses are 128 bits long. This larger address size allows for the generation of 3.4 × 1038 address values, which should be more than enough for current and future applications and eliminates the need for address conservation practices such as NAT (Network Address Translation) that IPv4 requires. IPv6 also supports end-to-end communication, enabling source and destination nodes to interact without intermediate systems such as NAT devices. This feature allows the development of new voice-over-IP, multimedia, and other types of network applications.

2.2 IPv6 Features

- Larger Address Space: IPv6 addresses are 128 bits long. This means 1564 addresses can be allocated to every square meter of this earth.
- Simplified Header: IPv6's header has been simplified by moving all unnecessary information and options (which are present in IPv4 header) to the end of the IPv6 header. IPv6 header is only twice as bigger than IPv4 provided the fact that IPv6 address is four times longer.
- End-to-end Connectivity: Since IPv6 does not use NAT, every device can directly reach other devices on the Internet, with exceptions like Firewall rules or organizations rules etc.
- Auto-configuration: IPv6 supports both stateful and stateless auto configuration mode of its host devices. This way, absence of a DHCP server does not put a halt on inter segment communication
- Faster Forwarding/Routing: Simplified header puts all unnecessary information at the end of the header. The information contained in the first part of the header is adequate for a Router to take routing decisions, thus making routing decision as quickly as looking at the mandatory header.
- IPSec: Initially it was decided that IPv6 must have IPSec security providing encryption and authentication right in the protocol suite itself. However, RFC-7872 shows that IPsec which employs extension headers result in packet drops when employed on Public Internet. The requirement to use IPsec was formally removed in the subsequent revisions of the IPv6 specifications (RFC6434).
- Extensibility: One of the major advantages of IPv6 header is that it is extensible to add more information in the option part. IPv4 provides only 40-bytes for options, whereas options in IPv6 can be as much as the size of IPv6 packet itself.
- Anycast support: IPv6 specification defines a new addressing scheme called "anycast address" that is an identifier for a set of interfaces (typically belonging to different nodes). A packet sent to an anycast address is routed to the "nearest" interface having that address, depending on the distance of the routing path. While 'unicast' means 'send to this specific member address' and 'multicast' means 'send to every member in this group', 'anycast' means 'send to any member in this group. The anycast mechanism can be used to implement the following:
    - The 'nearest' server selection or selecting the closest member of the group which enables the clients to communicate with the 'nearest' server having an anycast address.
    - Abstraction of service: It is troublesome to remember the IP addresses of servers to access services like DNS (or any other service like HTTP proxy in the Internet). If we can get DNS support for anycast addresses, we can access the nearest server with just the service name.

- Reliability: Since anycast address can be assigned to servers across the internet, if one of the servers is down, the service will not be impacted since packets with anycast address can be routed to the nearest server.

2.3 IPv6 Header

The IPv6 header structure is described below and shown in Figure 3.

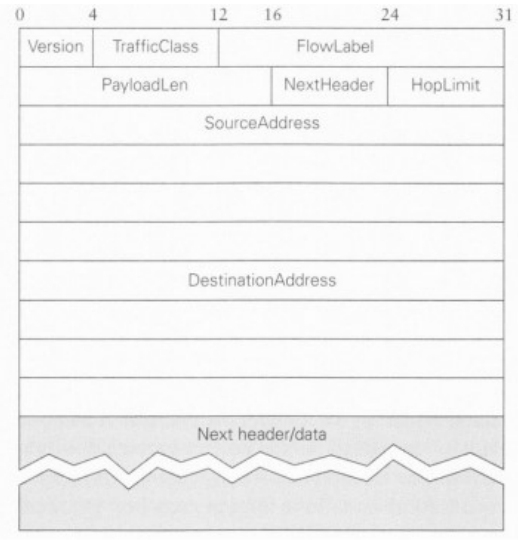


Figure 3: IPv6 Header

- Version: The Version field is set to 6 for IPv6.
- TrafficClass: The TrafficClass field identifies the priority and class of service of this packet.
- FlowLabel: The FlowLabel field is for future use in identifying packets that are part of a unique flow, stream, or connection
- PayloadLen: The PayLoadLen field defines the length in octets of the packet that follows the IPv6 header.
- NextHeader: The NextHeader field identifies the type of header that follows the IPv6 header. This replaces the Options and Protocol field of IPv4.
- HopLimit: The HopLimit field is a counter for the number of remaining hops the packet can traverse. This is simply the TTL of IPv4 renamed.
- SourceAddress: The IPv6 address of the node that originated this packet.
- DestinationAddress: The IPv6 address that this packet is destined for.

3 Security attacks in IPv6 Networks

When comparing IPv6 with IPv4, it is easy to say that the increased complexity introduces more attack vectors – more possible ways to perform attacks. However, while discussing the security of the protocol (with respect to its specifications), a more practical approach would be to ask how secure are the IPv6 deployments in comparison to the IPv4 deployments. The security of IPv6 deployments depend on the following factors:

- Confidence in the protocol
- Support for security analyzers and tools.
- Protocol maturity.
- Implementation/maturity of new specifications.

3.1 Reconnaissance attacks

In general terms, a reconnaissance attack is a knowledge gathering attack. It is used to gather the vulnerabilities of a host within the network. For example, an attacker might gather information regarding, the number of hosts, the numbers of open ports and system vulnerabilities of the target network. In IPv4 a simple ping sweep attack itself can give the attacker the foundation for the vulnerabilities and attacks in the network. However, such address scanning techniques are exhaustive in IPv6 networks with a huge address space. This doesn't mean that IPv6 are free from such attacks. Rather, some of the recent studies indicates that reconnaissance attacks in IPv6 are possible with attackers targeting public DNS servers for collecting IPv6 addresses.

Listed below are the two IPv6 DNS reconnaissance attacks.

a) DNS reverse zone scanning: DNS was not used for discovering IPv4 nodes since that was easily achievable by address scanning methods. Peter van Dijk [1] discovered that DNS reverse mappings could be used for discovering IPv6 nodes. In an IPv6 reverse lookup request each query is built by an IPv6 lookup address and the suffix 'ip6.arpa'. The attacker traverses the target 'ip6.arpa' zone by querying for the PTR records corresponding to the domain name. To extract IPv6 addresses from the 'ip6.arpa' zone, the users need to give a network prefix of the domain from the reverse DNS zone that they want to search. When the program receives the network prefix, it adds a new nibble (all the new nibbles start with zero) and appends it to the given domain name. The program then sends a reverse lookup with this new address block to DNS servers. If the response is NXDOMAIN, the program will increase its value for the current nibble and send the request again. If the response is NXERROR, the program receives this message, it adds a new nibble and appends it to the previous reverse query. Finally, if the response is hostname, the program stores the value to its list of IPv6 nodes.
b) DNSSEC Zone Reconnaissance: DNSSEC was introduced to circumvent the issues like DNS cache poisoning, DNS spoofing etc. Jakob Schlyter [2] identified an issue with the current DNSSEC validation approach. There is no cryptographic signature for a wild card query. If the DNS authoritative server cannot find a relevant response for a request, it can send an unsigned response indicating that there is no record for the request made. An attacker can easily forge such unsigned responses. NSEC (Next Secure Record) was introduced to resolve this security issue in DNSSEC. NSEC builds a trust chain with the existing domains and the type of records belonging to each domain. This enables the DNS authoritative server to respond to wildcard queries with a signed response. NSEC comes with a security issue in which the attacker can gather IPv4 and IPv6 addresses from the NSEC list. First, users need to give a domain name that they want to search. When the program receives the domain name, it appends a random string to the given domain name. If the domain uses the NSEC record, it will return a chain of existing zone names; the program then sends a lookup query to search IPv4 address and IPv6 address records for each zone.

3.2 Extension Header (EH) attacks

Since routers headers are allowed in IPv6 packet structure it is possible for an attacker to create packets with routing headers to reach hosts that normally would not accept the attacker's traffic. Further, if an end point accepts these headers and follows their routing instructions, trusted nodes could forward malicious packets or the flow of packets could lead to resource exhaustion at the routers, resulting in a DoS attack.

3.3 Neighbor Discovery Protocol (NDP) attacks

IPv6 uses Network Discovery Protocol (NDP) to find the MAC addresses to communicate with hosts in a Local Area Network. NDP is stateless and it doesn't require authentication. The traditional spoofing attacks for exploiting the IP to MAC resolution using ARP in IPv4 are also relevant in NDP. By using spoofed MAC addresses, a malicious host can also launch Denial-of-Service (DoS), Man-in-the-Middle (MiTM) attacks etc. in IPv6 network. Although there are various detection/prevention mechanisms available for IPv4, many of them are not yet implemented in IPv6 as the protocol is relatively new and slowly coming in use.

3.4 Packet amplification attacks

DoS attacks can be launched on a multicast group by sending messages to the group address requesting the members to leave. Since IPv6 relies on multicast and eliminates broadcast, this kind of attack can seriously impede a node's operation. IPv6 has reserved multicast addresses like ff01::1, ff01::2, ff02::1, ff02::2, ff05::2, which specifies "All nodes in interface-local"/"All Routers in interface-local"/"All nodes in link local"/"All Routers in link local"/"All routers in site local" respectively, it is easy for an attacker to alter messages directed to these addresses on a network and receive information that helps identify nodes on which to target attacks.

3.5 DHCPv6 attacks

A DHCPv6 client configures its IP address and other network parameters based on the information contained in DHCPv6 server messages it received. However, the client does so without first verifying the legitimacy of the message source. Therefore, attackers that are connected to the same link-local network, could masquerade as legitimate server and inject fake messages into the traffic to fool the client. The attack occurs when the client sends a Solicit message asking the server to reply. An attacker on the network will respond back with a fake Advertise message containing wrong network configuration parameters. Since the client does not have at its disposal a mechanism to verify the source of this message, it will readily accept the message and configure its IP address, as well as other network parameters with incorrect information. Hence, the client is under attack such as DoS or redirect the user to rogue servers as shown in Figure 4. Accordingly, authentication of the DHCPv6 server message is considered essential in IPv6 networks.

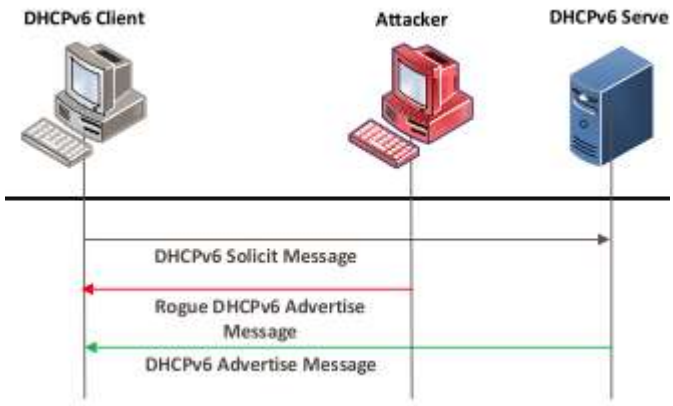


Figure 4: DHCPv6 Attack -Using Fake Advertise Message

3.6 DOS attack on DAD protocol

A DoS attack on an IPv6 network can be launched by exploiting vulnerabilities in the DAD (Duplicate address detection) procedure.

For this, an attacker on the local link waits until a node sends an NS packet. The attacker falsely responds with a neighbor advertisement packet, informing the new node that it is already using that address. Upon receiving the NA, the new node generates another address and repeats the DAD procedure; the attacker again falsely responds with an NA packet.

3.7 Security attacks due to the IPv6 Transition mechanisms

IETF has developed three major transition mechanisms for IPv6.

- Dual-stack: The nodes have two protocol stacks (IPv4 and IPv6) enabled and use IPv6 to contact IPv6 nodes and use IPv4 to contact IPv4 nodes.
- Tunnels: Hosts or routers send and receive IPv6 packets using an overlay network of tunnels established over an IPv4 network or over label switched path (LSP
- Protocol translation: A protocol translator acts as an intermediary between the IPv4 and IPv6 worlds.

Some of the security vulnerabilities of running dual-stack are:

- Protected against IPv4 attacks but not IPv6 attacks. A lot of users are not aware that their operating system is running both version of the protocol automatically.
- Denial of Service attacks

A list of vulnerabilities of running tunneling:

- Address spoofing: Most of the tunneling protocols like 6in4 has no security features, thus one can inject IPv6 packets by spoofing the source IPv4 address of a tunnel endpoint and sending it to the other endpoint.
- Reflection attack: Even in an environment where tunnels are properly implemented, it is still vulnerable to threats such as DoS attacks, Reflection DoS attacks and also Service Theft. This is when a malicious node, site or an operator uses the service without authorization.

4 IPv6 Security Solutions and Tools

4.1 Security from protocol perspective: Secure neighbor discovery protocol (SEND)

SEND adds three additional features to NDP:

- address ownership proof
- message protection
- a router authorization mechanism

To achieve these enhancements, SEND comes with four new options (CGA, RSA signature, nonce, and Timestamp) and two ICMPv6 messages for identifying the router authorization process.

Cryptographically generated address (CGA): CGA option carries the associated CGA parameters so the receiver can validate the proper binding between the public key (used to verify the signature) and the CGA. The CGA is an essential part of SEND, proposed to prevent address stealing. It authenticates IPv6 addresses without requiring third-party or additional security infrastructure. CGAs are IPv6 addresses, in

which a one-way hashing of the node's public key and other auxiliary parameters generates an IID. Thus, a node's IPv6 address is bound to its public key. The receiver can verify this binding by recomputing the hash value and comparing it to the sender's IPv6 address's IID.

RSA signature: SEND uses the RSA signature option to authenticate the sender's identity. Initially, each node must generate or obtain a public/private RSA pair before it can claim an address. The sender signs the outgoing messages with the private key, which corresponds to the public key used in CGA's generation algorithm. This signature prevents attackers from spoofing CGA addresses.

Nonce: The nonce option uses a random number to ensure that an advertisement is a fresh response to a node's solicitation. SEND includes a nonce option in the solicitation message and requires advertisements to include a matching option. This prevents a replay attack in solicited messages, such as NS/NA and RS/RA, which can be used for two-way communication but not for one-way communication messages.

Timestamp: SEND uses the Timestamp option to ensure replay protection against unsolicited advertisements, such as periodic RAs and RMs. Here, the assumption is that all nodes have synchronized clocks, so the node can prevent replay attacks by carrying out a time-stamp-checking algorithm.

Router authorization: SEND uses authorization delegation discovery (ADD) to validate and authorize IPv6 routers to act as default gateways and specifies IPv6 prefixes that a router is authorized to announce on its link. ADD relies on an electronic certificate issued by a trusted third party. Before any node can accept a router as its default, the node must be configured with a trust anchor that can certify the router via certificate paths. So, the node requests that the router provide its X.509 certificate path to a trust anchor, which is preconfigured on the node. The router shouldn't be trusted if it fails to provide the path to the trust anchor.

However, SEND faces limitations in several areas, including computation, implementation, deployment, and security, which might prevent CGA and SEND use and leave NDP messages vulnerable to potential attacks.

4.2 Router Advertisement-guard (RA-Guard)

In an IPv6 deployment, routers periodically multicast Router Advertisement (RA) messages to announce their availability and convey information to neighboring nodes that enable them to be automatically configured on the network. RA messages are used by Neighbor Discovery Protocol (NDP) to detect neighbors, advertise IPv6 prefixes, assist in address provisioning, and share link parameters such as maximum transmission unit (MTU), hop limit, advertisement intervals, and lifetime. RA messages are unsecured, which makes them susceptible to attacks on the network that involve the spoofing (or forging) of link-layer addresses. Also, unintended misconfiguration by users or administrators might lead to the presence of unwanted, or rogue, RA messages, which can cause operational problems for neighboring hosts. IPv6 Router Advertisement (RA) guard can be configured to protect the network against rogue RA messages generated by unauthorized or improperly configured routers connecting to the network segment. RA guard works by validating RA messages based on whether they meet certain criteria, configured on the switch using policies. RA guard inspects RA messages and compares the information contained in the message attributes to the configured policy. Depending on the policy, RA guard either drops or forwards the RA messages that match the conditions.

4.3 Tools for testing IPv6 networks

Deploying IPV6 in home networks or testing security solutions can be challenging since there are not many tools which offer the full capability to test the various features/attacks in IPv6 networks. However, the following is a list of tools that we can use while testing security in IPv6 networks:

- Neighbor Discovery Protocol Monitor (NDPmon): monitors the local network and reports any suspicious ND messages.
- ddaddos: monitors a network to detect any DAD-based attack
- Nmap: network vulnerability scanner

- Snort: intrusion detection and prevention system
- Wireshark: network protocol analyzer
- Netcat6: utility to read and write data across IPv6 network connections

5 Building a secure home network with IPv6

We have discussed the various attacks possible in an IPv6 network and some of the solutions like SEND that can be used to improve the security. However, most of these solutions are evolving and we are dependent on the ISPs to really adapt to such security best practices. With new gadgets seeking to make homes smarter and more efficient, it is important to learn how to secure the connected devices throughout your smart home. The Internet of Things/IoT have created new opportunities for cybercriminal to infiltrate into home networks. Consider a few scenarios:

- An attacker gaining access to the baby monitor for spying.
- An attacker gaining access through an IoT device for a ransomware attack thereby demanding a ransom to get the user's systems working again.
- An attacker altering the firmware update URL of a smart thermostat to his system's web address to understand when the home user is away from home.

All these scenarios underline the fact that securing home networks is as equally important as to automating them. IPv6, with its complexity, adds new attack vectors to smart home networks. It might be possible that a smart home user has adopted good security practices but is not aware whether these practices are enough if IPv6 is deployed in his/her network. The following section will evaluate some well-known security practices against their IPv6 compatibility.

5.1 Security practices for a smart home network with IPv6

Let us evaluate a regular smart home network (Figure 5) where the user has installed/deployed security software or followed some of the known security practices. We will detail how deployment of IPv6 can open such networks to security vulnerabilities and explain some prevention methods to make IPv6 space more secure.

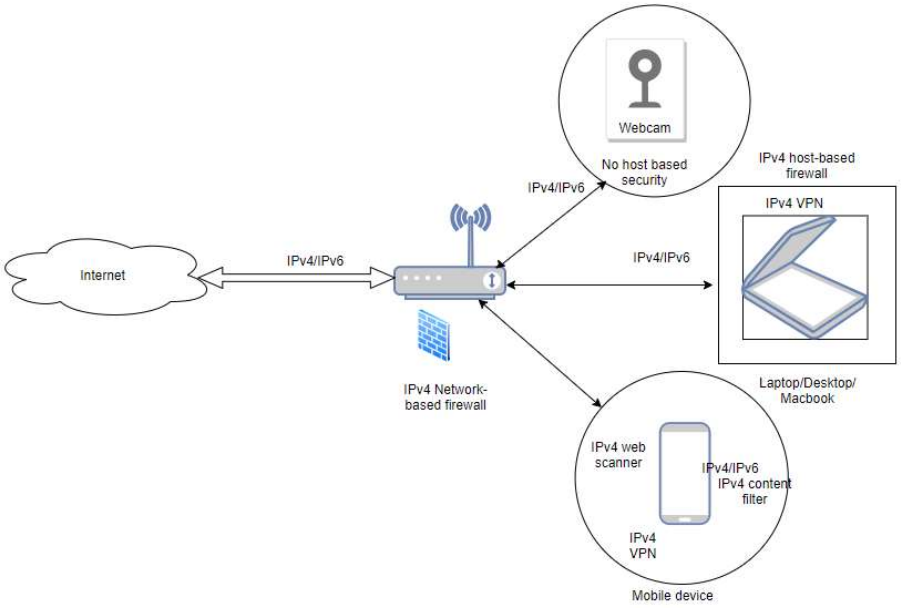


Figure 5: Smart Home Network with IPv4 Security Practices

5.1.1 Virtual Private Network

VPN (Virtual Private Network) is software that helps to connect smart home devices securely. This software encrypts the online data by passing it through a VPN tunnel thus making the data untraceable to anyone who is trying to snoop outside the network. A VPN masks the real IP address of a device and gives a fake one so that no one can trace the user's internet traffic thereby making the user's traffic invisible online. Installing VPN in mobile devices like Laptops/Macbooks/iPhones/Android devices are important since these devices might connect to public internet in cafes or airports etc.

However, Most VPN providers do not support IPv6 yet. If a VPN service doesn't support IPv6 and if the device is connected to one of its servers, then one of two things will happen.

- The VPN will successfully block all IPv6 traffic and force everything to use IPv4 where it will be protected, or
- the IPv6 traffic is not blocked and any web searches which use IPv6 will go via your ISP and not through the encrypted VPN tunnel. Also, most browsers, apps and operating systems are designed to prefer IPv6 if that is available. This means that the user might see his/her VPN app is connected and running, however all their IPv6 traffic is exposed online.

It is important for the smart home users to check the following before purchasing a VPN software for their devices:

- Complete IPv6 support: The VPN should be IPv6 supported. This means that the VPN has servers which can be contacted using IPv6 addresses or can work with translated IPv6 addresses.
- Dual stack compatibility: If the VPN does not support pure IPv6, then it should be dual stack compatible. This means that the VPM will block all the IPv6 traffic and fall back to IPv4 for user's traffic. Though this will mean that user will not be able to use IPv6, it will ensure that the online traffic is router via the tunnel.
- VPN on home gateway: There are services which provides VPN on the home gateways. This is a boon, especially for a home network with a lot of IOT devices. Installing VPN in IOT devices is not straightforward and having a VPN enabled router ensures that the security and privacy offered by the VPN can be extended to every device in the home network including the virtual assistants, security cameras, smart thermostats, baby monitors and smart lights. If the user decides to invest in this security solution it is equally important to check with the vendor regarding the solution's IPv6 support. For VPN on home gateways, it is better to go with a pure IPv6 supported solution than a dual stack supported solution since this will help the user in understanding how their ISP deploys IPv6, whether it is dual mode, it is using IPv4 networks to tunnel IPv6 packets or if any translation methods are used like NAT64 or 464XLAT etc.

5.1.2 Host based firewalls

A firewall is a security device (computer hardware/software) that can protect a networking device like Laptops/Gateways etc. by filtering traffic and blocking outsiders from gaining unauthorized access to the private data in the device. Firewalls can be either host based, or network based. A host-based firewall is a piece of software that runs on individual devices connected to the home network. These types of firewalls are a granular way to protect the individual hosts from viruses and malware, and to control the spread of these harmful infections throughout the network. In a smart home network, the user can go with two types of host-based firewalls, mainly the one that comes with the operating system (like Windows PC/Macbook/Linux firewalls) or a firewall security software tuned to the user's networking infrastructure.

While using firewalls to secure home networks it is important for the smart home user to understand that some firewalls cannot handle IPv6 traffic at all, others are able to handle it but have limited abilities to filter IPv6 traffic, and still others can filter IPv6 traffic to approximately the same extent as IPv4 traffic. Following points should be considered before configuring firewall rules in the home devices:

1. The firewall should be able to use IPv6 addresses in all filtering rules that use IPv4 addresses.
2. The firewall needs to be able to filter ICMPv6, as specified in RFC 4890.

3. The firewall should be able to block IPv6-related protocols such as 6-to-4 and 4-to-6 tunneling, Teredo, and Intra-site Automatic Tunnel Addressing Protocol (ISATAP) if they are not required.
4. The firewall should be able to block remote access to IoT devices where it is not necessary. Devices like thermostats or baby monitors might need remote access, but not all of them really need them.

Among the points listed above, points (1) and (2) can be easily achieved by a smart home user if the administrative interface of the firewall is intuitive or maybe even with a little help from the instructions in OS/security software's manual. However, blocking tunnels in point (3) is where it gets tricky. Tunneling methods like Teredo can infest even smart home users with an ISP that has only IPv4 networks but most of the devices in the home are IPv6 enabled.

Teredo operates using a platform-independent tunneling protocol that provides IPv6 connectivity by encapsulating IPv6 datagram packets within IPv4 User Datagram Protocol (UDP over port 3544) packets. Teredo routes these datagrams on the IPv4 Internet and through NAT devices. Teredo nodes elsewhere on the IPv6 network (called Teredo relays) receive the packets, un-encapsulate them, and pass them on. uTorrent (Figure 6) runs very well over Teredo, and that the BitTorrent community is discovering IPv6 as a way of avoiding network congestion controls that are used by ISPs to manage BitTorrent traffic on IPv4 networks.

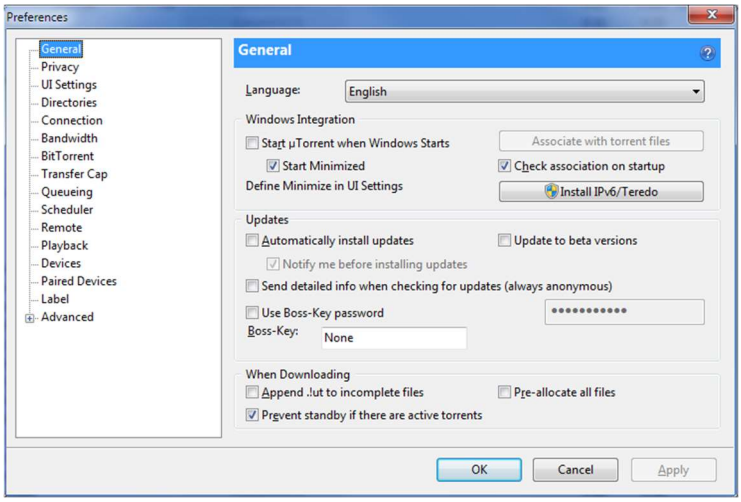


Figure 6: Teredo Tunnel Option in uTorrent

Several Bit Torrent trackers started taking advantage of Teredo Tunnels and would work with the configuration of Teredo. What this would do is allow the Torrent user to bypass the firewall security mechanisms for blocking Torrent users. Some clients would use encrypted UDP over known ports (like DNS) and could exit a protected network by appearing to be harmless. The IPv4 network becomes a transit for these tunnels off to the DarkNet, where IPv6 relays provide anonymity and evasion from detection mechanisms. Malware can also take advantage of the same overlay network mechanisms and go undetected due to lack of IPv6 security implementations.

Host-based firewalls might prove less helpful due to the following reasons:

- The smart home user never bothers to enable settings beyond the lax default settings of his devices.
- Puts too much responsibility on the homeowner.
- The homeowner needs to review each of their connected devices, identify the ports the applications listen on and connect to, and configure their firewall accordingly with nuanced understanding of their network.
- Cannot configure rules for headless IOT devices.

Nevertheless, with more attack vectors surfacing, it is critical that the smart home users ask the right questions before configuring host-based firewalls to protect their individual devices.

5.1.3 Network based firewalls

While host-based firewalls are a good way to block unwanted/malicious traffic, there might be a lot of devices in a smart home network which cannot be configured with a host-based firewall. Network-based firewalls are best suited to protect personal devices like wearable fitness gadgets and many other IoT devices that connect to the wireless networks.

Network firewalls filter traffic going to and from the internet to secured local area networks (LAN). These can be configured by the homeowner by logging in to the home gateway's web management console or by purchasing a security solution that is installed in the home gateway by the ISP. Either way, a network-based firewall is essential to a smart home network and the homeowner must ensure the following rules are enabled.

- Disable all port forwarding rules for IPv6: Ensure that popular ports that are used for SSH, Telnet or FTP are disabled to be accessed from outside the network. It might be possible that these rules already exist for IPv4 but ensure that they are available for IPv6 traffic as well.
- Disable access to the router's management console via IPv6 address: This might be enabled by default for all access from WAN (wide area network) segment, but it is important to restrict the access to the router's management console via the homeowner's devices except for one device (maybe the homeowner's personal desktop or laptop). Attackers can gain access to the user's router by simply gaining access to one of the least protected devices in the network. Also, while restricting access, rules should be equally applied to the IPv6 web address of the management console.
- Enable Bogon filtering for IPv6 Bogon addresses: Bogons are the IP address ranges that either have not been allocated or are reserved. The bogons list originated from RFC 3330's list of "Special-Use IPv4 Addresses," and now a similar list of "Special-Use IPv6 Addresses" is documented in RFC 5156. Packets with these addresses, either used as source addresses or destination addresses, should not be routed on the Internet. These are often blocked at IPv4 routers explicitly because there are a finite number of these. In general, packets that are sourced from bogon addresses should be filtered. While this is a good goal, it also means that the rules need to stay on top of the allocations as they are made and adjust the filter lists accordingly. These bogon lists can change several times each year. Enabling Bogon filtering might be as simple as clicking on a toggle button in the router's management console or as difficult as the homeowner to configure these rules by themselves. If the homeowner is not geeky enough it is always good to go for a security solution that come enabled with these configurations.

5.1.4 Next Generation Firewall (NGFW)

While a home gateway has a built-in firewall, it may not prove to be sufficient since it might lack important security features like,

- Intrusion Prevention System (IPS)
- Malware protection,
- Content filtering,
- SSL/SSH interception,
- QoS management

A next-generation firewall (NGFW) is an integrated network platform that combines a traditional firewall with other security functionalities such as the ones just mentioned. An NGFW has all the capabilities of a traditional firewall as well, making it powerful in detecting and protecting against cyberattacks.

Recent studies show that 5% of the CCTV Botnet was using IPv6. When the Internet of Things (IoT) fully adapts to IPv6, we will begin to see a rise in DDoS botnets leveraging IPv6. In the Mirai Botnet, IPv6-enabled devices were seen in the attack matrix sending out floods. The Mirai Botnet has reached over 1 Tbps in size for flooding attacks. That same 5% of IPv6 hosts could essentially produce 50Gbps of DDoS flooding.

Next Generation Firewall will be a key factor in protecting smart home networks from such botnet attacks. The NGFW investigates 'unknown' traffic in the home network by:

- Track source and destination, volumes of traffic
- Correlate against URL, IPS, malware and file transfer records
- Investigate "unknown" traffic for potential unauthorized user behavior or potential botnet behavior.

NGFW can maintain and update the blacklist IP addresses that are classified as botnet IPs. Any traffic to these addresses will be blocked by the firewall rules.

Next-generation firewall is an expensive investment, but the level of security boost for the smart home makes it a worthy investment. In case of NGFW in home networks with IPv6 enabled devices, a homeowner should make sure of the following before purchasing the solution.

- Botnet blacklist is updated for IPv6 addresses: NGFW in home networks should update the blacklist botnet IP addresses not only for IPv4, but for IPv6 as well. Otherwise, though the firewall is protecting IPv4 botnet attacks, the attacker might gain access to the home network using IPv6 traffic.
- Content filtering that supports IPv6: This is especially important if the homeowner has parental controls in place. Blocking a site for the child devices should mean that it is blocked for both IPv4 and IPv6 addresses. Otherwise, while the parent might have blocked all websites related to drugs, the child might be able to gain access to the content if some of the sites have IPv6 addresses.
- SSL /SSH interception for inbound IPv6 traffic: Inspecting SSL traffic to check the type of traffic should apply for both IPv4 and IPv6. This will help in updating the NGFW with respect to the type of traffic and its reputation.

5.1.5 Separate network for IoT devices

Many modern routers provide the ability to set up a secondary network. By creating a separate network dedicated to the IoT devices, the main network can be safeguarded against IoT threats. This means relatives, friends, and guests can log into a network that isn't related to the IoT devices. So, the local smart home network is only accessible by the homeowner (and his/her family). As putting IoT devices on a different network keeps them detached, if hackers do manage to get through, they can't access any of the more important devices, such as the user's laptop or smartphone.

A homeowner can create multiple LAN segments like:

- Office Network: All important devices where the homeowner stores personal data like bank accounts or legal papers can be connected to this segment.
- Home IoT Network: All devices needed for a smart home automation like smart thermostat, smart speakers, fitness gadgets etc can belong to this segment.
- Guest network: All the devices of the guests can be connected to this segment.

So, what does creation of multiple LAN segments mean in an IPv6 home network? Below are some of the points that the homeowner of a smart home network with IPv6 should follow:

- Ask for a /56 or /48 prefix: Some ISPs offer an /64 prefix. With a single /64, the home gateway will have just one possible network on the LAN side, and it will not be possible to subnet, assign VLANs, alternative SSIDs, or have several chained routers in the customer network, etc. So, creation of multiple LAN segments should begin by a request for the /56 or /48 prefix.

- Assign firewall rules to each LAN segment: Make sure that the network-based firewall rules are customized to each LAN segment. Also, ensure that after segmenting the LAN, devices from each segment cannot access the devices in the other segment.

This section has listed some of the security practices used in smart home networks that should be adapted to IPv6. Bottom line is that, a smart home network is as secure as its least secure device. So, while automating home might make life easier, it is equally important to secure the automation as well. Along with the above-mentioned practices, the user must follow the below security practices regardless of IPv4/IPv6 networks:

- Change the home gateway's Default Name
- Set a unique password
- Use the highest level of encryption like WPA2 or WPA3
- Disable features in IoT devices that is not used
- Keep the devices updated by frequently upgrading
- Keep the gateway secure by doing frequent firmware upgrades

Table 1 below details shows three broad categories of devices found in a home network and the security solutions that can be used to secure them mostly from an IPv6 perspective.

	VPN	**Host based firewalls**	**Network-based firewalls**	**NGFW**
IOT Devices Thermostat/Baby monitor/Alexa/Google Home	Cannot be installed Can connect to VPN enabled gateways with IPv6 support	Most IoT devices don't have the support Need to add IPv6 firewall rules if the device supports	Must for all IoT devices since endpoint protection is rare. Clone IPv6 rules for the existing IPv4 rules	Enable IPv6 blacklist for botnet Enable IPv6 SSL interception
Office Devices- Laptop/Desktop/Macbooks	Must be installed. Use IPv6 supported VPN software	Must be configured. Clone IPv6 rules for the existing IPv4 rules Configure rules to connect to only know tunnels	Must for all office devices to disable port forwarding via SSH/Telnet. Clone IPv6 rules for the existing IPv4 rules	Enable IPv6 blacklist for botnet Enable IPv6 SSL interception Enable content filtering
Mobile Devices (connected to LTE) iPads/iPhones/Android phones/Tablets	Must be installed. Use IPv6 supported VPN software	Cannot configure host-based firewall rules. Can install endpoint security solutions like Web scanner, web protection, content filtering software etc. Need to install endpoint security apps with IPv6 support.	Must for all smartphones since smartphones could have been vulnerable since they move in-out of the network. Clone IPv6 rules for the existing IPv4 rules	Cannot apply rules if the device moves out of the network. Enable IPv6 blacklist for botnet while inside the network. Enable IPv6 SSL interception while inside the network Enable content filtering while inside the network

Table 1: IPv6 security solutions in smart home devices

6 Conclusion

IPv6 transition is an interesting topic in the networking world. Though methods are in place to protect enterprise networks from IPv6 attacks, it is equally important to focus on the security challenges that it brings in the home networking space. When new functionality is developed, security is an afterthought. IPv6 solved the immediate problem of exhausting the address space but it also introduced new security vulnerabilities while creating gaps in many IPv4 monitoring and security tools. ISPs don't give home users a choice about moving to IPv6 and rarely provide security guidelines that a home user can understand. Now that IPv6 functionality is upon us, typically in a dual stack, ISPs and security vendors need to up their game for IPv6 security. With more focus from security vendors and ISPs in solving these issues for the home space, the transition to an IPv6 global era will be safer.

References

1. P. van Dijk, PowerDNS, P.Hoffman, "Common Features for Encrypted Recursive to Authoritative DNS draft-pp-dprive-common-features-02", https://www.ietf.org/id/draft-pp-dprive-common-features-02.txt

2. Joe Abley, Jakob Schlyter, Guillaume Bailey, Paul E. Hoffman, "DNSSEC Trust Anchor Publication for the Root Zone", RFC 7958: 1-14 (2016)

3. IPv6 Task Force, "Technical and Economic Assessment of Internet Protocol Version 6 (IPv6)," Jan. 2006, US Dept. of Commerce; www.ntia.doc.gov/ntiahome/ntiageneral/ipv6/final/ipv6finalTOC.htm.

4. G. Van de Velde et al., Local Network Protection for IPv6, IETF RFC 4684, May 2007; www.rfc-archive.org/getrfc.php?rfc=4864&tag=Local-Network-Protection-for-IPv6.

5. S. Convery and D. Miller, "IPv6 and IPv4 Threat Comparison and Best-Practice Evaluation (v1.0)," Mar. 2004; www.seanconvery.com/v6-v4-threats.pdf.

6. P. Nikander, J. Kempf, and E. Nordmark, IPv6 Neighbor Discovery (ND) Trust Models and Threats, IETF RFC 3756, May 2004; www.rfc-archive.org/getrfc.php?rfc=3756.

7. T.Doan, "IPv6 Security Assessment," 1 June 2006, SAIC

www.hpcmo.hpc.mil/Htdocs/DREN/2006SEP01_IPv6SecurityAssessment.pdf.

8. J. Arrko et al., SEcure Neighbor Discovery (SEND), IETF RFC 3971, Mar. 2005; www.rfc-archive.org/getrfc.php?rfc=3971.

9. Growth of Internet, "https://growthchart.weebly.com/"

10. RFC-7872, https://datatracker.ietf.org/doc/html/rfc7872

11. RFC-6434, https://datatracker.ietf.org/doc/html/rfc6434

12. RFC-4890, https://datatracker.ietf.org/doc/html/rfc4890

The Do's and Dont's of Accessibility

Michael Larsen

mkltesthead@gmail.com

Abstract

Accessibility is a large topic and one that often gets a variety of approaches to deal with. Often it is seen as having to focus on a large checklist (the WCAG standard) and make sure that everything complies. While this is a great goal and focus, often it is overwhelming and frustrating, putting people in the unfortunate role of having to read and understand an entire process before they feel they can be effective.

My goal is to help condense this a little and give some key areas to focus on and be effective in identifying Accessibility issues quickly and helping testers become effective advocates. We will look at ways to find issues, advocate for them and help make strides to greater understanding and focus moving forward. We can use a little to provide a lot of benefits.

Biography

Michael has worked on a broad array of technologies and industries including virtual machine software, capacitance touch devices, video game development, and distributed database and web applications. He currently works with PeopleFluent, located in Raleigh, NC, USA. He writes a software testing blog called TESTHEAD (http://mkltesthead.com/).

Michael served as a member of the Board of Directors for the Association for Software Testing from 2011-2015. He was their Treasurer and then their President. Currently, he helps teach their Black Box Software Testing classes. Michael is also the current producer and a regular commentator for The Testing Show, a podcast produced for QualiTest (available in Apple Podcasts, Google Podcasts, and Spotify).

1. Introduction

Accessibility can be defined as “The design of products, devices, services, or environments for people with disabilities. Accessibility allows for design compatibility with a person’s assistive technology”.

Accessibility utilizes the WCAG standard and other regulatory documents as required in various counties to determine that our software complies with suggestions and regulations. The WGAG standard is large and getting larger all the time. It is understandable that people would look at the checklist, feel deflated, and wonder if they would ever be able to understand it all, much less actively use it.

To this end, I want to share some ideas based on key areas and help people get their hands dirty quickly, get some quick wins, and develop skills that will help interested testers and developers approach Accessibility more effectively.

2. Why Do We Need to Deal with Accessibility?

If we are lucky enough to live a long and complete life, we are likely to experiences a primary or secondary disability of some sort: visual, auditory, motor, cognitive, or a combination of any of those listed.

Nearly 1 in 5 people have some form of disability in the USA. More than a billion people in the world today experience some form of disability. Even those who don’t consider themselves disabled have some sort of disability – even if it is simply wearing glasses. It makes social and economic sense to plan for a better user experience by incorporating design philosophies that address Accessibility.

3. Situational Disabilities

Persistent issues (low vision, low hearing, limited mobility, cognitive disability) are considered primary or chronic disabilities. However, there are situations people without chronic disabilities face. We refer to these as “situational disabilities”.

Examples of situational disabilities include:

- Background noise (hearing)
- Distracted tasking (cognitive)
- Small text/non-scaled web page (vision)
- Foreign language (literacy)

Have you been in a country where you can’t read the language or even the writing? This is akin to the frustration people with disabilities feel using systems not designed for Accessibility.

One of my favorite examples of Accessibility and Inclusive design is IKEA’s assembly instructions for any product. Pictographs and images overcome literacy issues. Also, in most cases, only one set of instructions is needed, rather than repeating the same series of instructions multiple times in several languages. This likewise cuts down on the cost of repetitive printing, which has a bottom line effect as well as a customer experience improvement.

4. Ten Principles of Web/Mobile Accessibility

Jeremy Sydik's book "Design Accessible Web Sites" focuses on "Ten Principles of Web Accessibility". Those ten principles are:

1. Avoid making assumptions about the physical, mental, and sensory abilities of your users whenever possible.
2. Your users' technologies are capable of sending and receiving text. That's about all you'll ever be able to assume.
3. Users' time and technology belong to them, not to us. You should never take control of either without a really good reason.
4. Provide good text alternatives for any non-text content.
5. Use widely available technologies to reach your audience.
6. Use clear language to communicate your message.
7. Make your sites usable, searchable, and navigable.
8. Design your content for semantic meaning and maintain separation between content and presentation.
9. Progressively enhance your basic content by adding extra features. Allow it to degrade gracefully for users who can't or don't wish to use them.
10. As you encounter new web technologies, apply these same principles when making them accessible.

These principles are helpful when it comes to designing applications and they are also helpful when it comes to framing how they should be tested.

5. Mobility Issues

Mobility issues are often the most visible of disabilities. A person may have missing or immobile limbs or extremities, or they may have a loss of fine motor control. These conditions can be present from birth, but mobility impairments also are the most common secondary, or situational, disabilities. Injuries to the wrist or hand that require immobilization, such as bone breaks, can significantly affect the way people interact with their systems and how they input information.

A variety of conditions, such as multiple sclerosis, Parkinson's disease, diabetes, stroke, myelopathy, and arthritis, may make it difficult to control a mouse or a stylus, or otherwise effectively use a standard keyboard.

There are several approaches that can help an application be more accessible to those with mobility disabilities.

Types of mobility disabilities

Primary mobility disabilities include the following:

Paralysis: A variety of conditions can result in a lack of movement in the extremities. Birth defects such as spina bifida and cerebral palsy can have a significant impact on the ability of extremities to operate, with a gradation of minor fine motor control issues to full immobility.

Limb loss or absence: Meromelia is a birth defect where one or more limbs may be lacking a part or only partially formed. Diseases or injuries also can require the amputation of limbs and extremities.

Ensuring your application is usable by those with mobility issues

The goal for truly inclusive design is to engineer your application in such a way that assistive technologies aren't even always necessary. Here are a few suggestions, many of which do not require assistive technology, that you should check to ensure your application is accessible by those with mobility issues.

- Ensure all functions can be accessed with the keyboard

This is specific to web applications and regular computer systems. Interactions that focus heavily on mouse usage will be a challenge for people with a limited ability to move a mouse or to specifically control and target its pointing interface.

Making sure that keyboard options are available and clearly indicated will help considerably. It also helps to provide keyboard commands that are standard and do not require complex sets of keystrokes to execute.

- Use tabs and skip links to get to the main content

It's very helpful to those with mobility issues to have the ability to use the tab key to navigate, as is making the first tab item a skip link, so as to bypass the navigation elements, if necessary. Verify that the tab order is consistent with the way the material is presented, most often left to right and top to bottom.

- Test your product with a capacitive touch stick

Many users will need an alternative to keyboard input. Both with traditional computer systems and mobile devices, I use a touch capacitance pen or pointing device and hold it between my lips or teeth. The rubberized end acts as a pointer and a substitute for a finger to interact with a typical computer keyboard. It does have some limitations in that I can only press one key at a time, so using "sticky keys" and other time-based key combinations helps me interact with a system.

- On touchscreen devices, closing my lips around the barrel of the pen also allows me to use the human body's electrical connectivity to utilize the rubberized end on the screen.

This helps extend the ability of a classic touch stick to a smartphone or tablet and register the touch events the same way another user would with their finger.

- Drive your application with voice-recognition software:

For individuals with limited or no mobility in their arms or hands and where movement with a touch stick may be impractical, voice recognition software is a huge advantage. Voice recognition tools allow users to use their voice to execute commands, navigate sites, and fill in the information, as well as compose longer text entries.

I find spending time trying to navigate an application through a voice-recognition application to be both enlightening and frustrating. Commands for voice-recognition software are standardized to common keystrokes for typical computer actions, so making sure that my software can effectively respond to voice commands is important.

6. Vision Issues

When developers plan to make an application accessible to those with visual impairments, often the first thoughts are about screen readers and making sites and applications available to people who are blind. While that is an important aspect, it should not be where the discussion about accessibility stops. Vision is a broad area, and there are a number of accessibility issues in the area of vision. Here are some examples of primary visual disabilities:

Refractive error: Focusing light on the retina. This includes nearsightedness, farsightedness, and astigmatism. In many cases, glasses can remedy this, but some cases are too severe for glasses to help

Cataracts: A clouding of the lens in the eye that can be slight to severe

Macular degeneration: A condition that causes blurred or no vision in the center of the visual field

Color blindness: A situation where vision may be normal in most means but the ability to see certain colors is hindered

In addition, there are a variety of visual situational disabilities that many of us may deal with:

- Having our eyes dilated as part of a vision procedure

- Stepping outside into the bright sunlight and trying to see something on our phone or device

- Corneal abrasion: Think of getting sand or dust blown into your eyes to the level that you can't wear contact lenses for a few days

Each of the situational disabilities described above is temporary. Still, each of them would benefit from accommodation, and we should make the applications we develop accessible to as many people as possible.

Design Considerations

How can we make applications more accessible to people with visual impairments? Below are some design suggestions, many of which do not require any assistive technology.

- Use appropriate color contrast

Using a distinct contrast of colors is especially important when dealing with color blindness, but this design practice is helpful for everyone. The usual combination is dark text against a light background, but this can be reversed if using a dark background. The Web Content Accessibility Guidelines (WCAG 2.1 is the current recommendation) encourages a color contrast of 4.5:1 for medium-sized bold or unbold text and 7:1 for small, unbold text.

- Enable scalable fonts

By allowing the user to magnify and scale the font size for their needs, users can more easily read the text in a variety of situations.

- Put all the information on the page

If you have to navigate a site with a vision impairment, it's frustrating to need to download information as separate documents. In some cases, it's necessary, such as with binary files or installable executables, but if dealing with text, where possible, make it available within the page itself.

- Avoid using color alone to display meaning

Green means go, yellow means proceed with caution, and red means stop. For users with normal vision, these color cues are common and can communicate a lot of information quickly. But for a color-blind user, this meaning can be missed. There isn't a single type of color blindness, so where possible, use text or graphics to display and convey greater meaning than color.

- Make your pages as linear as possible

Having pages that flow from top to bottom in a single-column format is easier for people who need to increase magnification on a page. The text will align with the magnification level, and the pages will display similarly to how they will at normal magnification. Spreading the content out over multiple columns

or in a side-to-side format makes it difficult to see where the content is, and navigating the page could be confusing.

- Do not separate elements that go together

By grouping text boxes and buttons that relate to each other, we can make sure that context is not missed and that actions that need to be performed in a certain order are easy to interpret. Think of a form layout that needs to be magnified by 200 percent. If you can't see the button that goes with a particular text box, it's a good bet that your users will miss it, too.

Considerations for Screen Readers

While the modifications discussed above benefit everyone, there are some situations where assistive technology is needed. Screen readers are applications that take the text that appears on a screen and reads it aloud for the user. There are a number of techniques that can be used to help make this process easier, and as an added bonus, these additions can be checked with automated scripts.

- Describe images and provide transcripts for video

The alt tag is very helpful for displaying additional descriptive meaning for an element. Any image that appears on the screen that is meaningful to a user should have a descriptive alt tag. For a video, transcripts can go into greater detail about what is appearing on the screen.

- Make content easy to navigate

Set the tab order so that the user can more easily navigate to the most important information. The first item should be a link at the top of a page that says, "Skip to main content," and allows the user to bypass the navigation elements on the page.

- Use HTML5 options to make pages more structured

Use the options available through HTML5 such as article, nav, sidebar, etc. These labels give a much better idea of what is being navigated. Use div tags sparingly, and if they must be used, give an alt tag description that can provide context.

- Provide a keyboard-only option to complete a task

It is common to rely on a mouse or another pointing input device to complete actions, but for screen reader users, a mouse is useless. Navigation and interactions are done via keyboard. If a workflow can't be completed with just a keyboard, users who rely on screen readers will not be able to use your application.

- If there are links, make the link text descriptive

If a page has multiple links that say "Click here," it is easy to lose the context of which link goes where. Be descriptive with the link, and define specifically where it will take you.

- All images have an alt tag and have descriptive alt text

This is a simple first step and it's one that is easily handled by any tool that can examine a web page. If there is an alt tag, there needs to be alt text associated with it. Granted, not every image is going to be meaningful to the user: Some are purely decorative or are repeated inside of a page. If an image is part of the page flow and is relevant to the content being displayed, it is important to verify that there is alt text for these images. For images that are just there for a sense of page aesthetics, null tags can be created.

- Tables have "and" values

Tables are challenging for screen readers and for keyboard navigation. Check that the tables are properly descriptive so that they are easier to navigate and provide context for the data in each row or column. Search for and tags in each table and identify any instances where they are not used.

- Inputs have labels

When using forms, it's important to identify all of the controls and what they represent, especially for assistive technologies like screen readers. The <label> tag helps to present that control to the user effectively. Example: If a form is looking for an email address, and the input type is "text" with a name and id of "emailaddress", adding <label for="emailaddress"> will provide that extra level of context for a screen reader.

7. Auditory Issues

Discussions of accessibility for applications are usually dominated by visual considerations. Unless we are dealing with sound directly, it's common to think less about the audible aspects of our sites and applications. However, there are numerous audible components of applications that we can improve, and provide an experience that more people can enjoy whether they are dealing with partial or total hearing loss. The key point to remember is that if there is a way to make content perceivable to people who can't hear, we should take the steps to make that happen.

Primary Auditory Disability

Like in the previous section regarding visual disabilities, auditory disabilities can be separated into primary (persistent) and secondary (situational) issues. And as with visual loss or limitation, hearing loss or limitation can have various stages and causes. When I consider hearing loss, I try to break it up into ranges:

- **Mild hearing loss:** Difficulty hearing sounds lower than about 30 decibels. If there is background noise, it may be hard to understand someone speaking.
- **Moderate hearing loss:** Difficulty hearing sounds lower than 50 decibels. To put this in perspective, that's approximately the volume of a refrigerator running. A hearing aid is likely to be helpful in this kind of situation.
- **Severe hearing loss:** Difficulty hearing sounds lower than 80 decibels. To put this into perspective, that's the volume of an average washing machine while it is running, or the average alarm clock. At this level, hearing aids may not be helpful.
- **Profound hearing loss:** Difficulty hearing sounds lower than 95 decibels. That's about the volume of a passing subway train.

In addition, there are a variety of auditory situational disabilities that many of us may deal with:

If you are in a noisy restaurant or bar and someone calls you on the phone, you may not be able to hear the phone ringing, even with normal hearing, much less hold a conversation with a person due to the noise level in the room

Ear infections can happen to all of us as a result of sickness, and often they can have a profound effect on hearing

These situations, again, are temporary or situational, but they would require users to have an option to perceive the audible information in an alternative way. Below are a few considerations for testers to verify the ability of hearing-challenged people to perceive audible content, many of which do not require assistive technology.

- Are there text alternatives where possible?

If a video is being displayed, the application should provide closed captions. If presenting a podcast, there should be a text transcript of the content. Additionally, aim to keep the message straightforward, and don't veer into inside jokes or figures of speech that may not be able to be interpreted as written. This is good advice for audio content in general.

- Is the page design smooth and simple?

If there are places in the application where the content is meant to be heard, that should be clear to the user. Follow the suggestion above to make sure there is a way to provide text content for the audio.

- Are audio signals used alone as cues?

If an alert relies on a sound, the application should display a message as well; or, if on a mobile device, vibrate to alert the user.

- Are there a variety of ways to communicate?

In an age where texting seems the most common way to communicate, many organizations still default to using a phone to handle issues. Allow for other ways to let this communication happen.

- Is the content structured clearly?

Too much badly formatted information can be overwhelming for many people, and this is also true for auditory impairment. Well-structured information, with clear headers, bulleted or numbered lists, separation of content, and a lack of clutter can help make pages and applications easier to deal with.

- The alt tag isn't just for screen readers

The alt tag is the simplest and most straightforward option and can be used with many elements. It's typically used with images to describe through screen readers, but an alt tag can also be used with audio content to describe what it is. There are limitations to this approach, however, as this would not be appropriate for a long presentation.

- Use track and VTT files for longer text

If we take advantage of the video and audio tags inside of pages, we can also leverage the track tag to provide a text equivalent of the audio that is being presented. The track tag takes a variety of options. Common examples include kind (what the track tag represents), src (the file that contains the content saved as a vtt file), and srclang (which defines the language/locale to use, such as "en", "de", "ja", etc.). If track files are used with audible content, this makes for an easy test to confirm their existence and verify that the breadth of language options we offer are being represented.

- Using the transcript tag to identify transcripts

For truly large text files, the transcript tag is still considered one of the best options. However, the ability to associate a transcript with an audio and video file depends on how the pages and content are structured. A newer feature in HTML5 allows for extending the tag to include transcript as an option. Example: Within a video tag, if the application references a video file, you can extend the track definition to include the kind "transcript" and make a link to a document. The link can be on the same page or it can be in an external document if desired. Once the "kind" transcript is associated, look for the transcript tag as part of a test to verify that it has been included along with the video file. If the goal is to make sure that the content you have worked so hard to create can be shared with (and purchased by, in many cases) the broadest number of people, it makes business sense to allow for multiple options to perceive that information. Just because your application's users can't hear the audio content doesn't mean they would not benefit from it. By giving an alternative and, hopefully, comparable experience, your sites and applications can be useful for more people.

8. Cognitive Issues

Cognitive disabilities cover a variety of conditions and experiences, but because they are not always obvious, they often get overlooked and are perhaps the least understood disabilities.

When it comes to addressing auditory, mobility, or visual issues through your website, the approaches toward accessibility are more straightforward. For visual issues, a screen reader can help. For auditory issues, using closed captioning can replace audio output. Mobility issues allow for voice operation or other methods to enter information and interact. But due to the number of possible causes for cognitive

disabilities, coming up with a set of solutions is challenging. And the goal for truly inclusive design is to engineer your site in such a way that assistive technologies aren't always required.

Bearing both of these factors in mind, there are several approaches that a tester can look for to verify that sites are accessible by those with cognitive disabilities.

Types of Cognitive Issues

As with visual or auditory impairment, there are primary and secondary, or situational, cognitive disabilities. Primary disabilities include the following:

Down syndrome: A congenital disorder that comes from having all or part of an extra copy of chromosome 21 (trisomy 21) that involves both physical and cognitive abnormalities

Autism: A developmental disorder frequently manifested by challenges with communication, forming relationships, and understanding concepts that are not literal

Asperger syndrome: An autism spectrum disorder that appears through difficulties with nonverbal communication and social interaction and frequently manifests a need for repetitive patterns of behavior

Dementia: A general term for a variety of symptoms, typically resulting in a decline I memory and a change to a previously normal personality and reasoning

Dyslexia: A disorder that makes learning to read and interpret words difficult, manifesting in mixing up letter or word order

Dyscalculia: A difficulty in understanding arithmetic, numbers, and mathematical concepts

There are also a variety of examples where cognition can be situationally impacted. These secondary disabilities could include the following:

- Difficulty when having to interact with a foreign language, particularly one that uses a different alphabet

- Being in a distracted, stressed, or emotional state

- Trouble understanding when wading through dense, minimally formatted & punctuated text

Here are a few considerations for testers to verify that an application is accessible by those with cognitive disabilities. Note that many of these do not require assistive technology.

- Is content structured with clear headings, sections, paragraphs, and lists?

Too much information badly formatted can be overwhelming for many people, and this is especially true for those dealing with cognitive issues. Create well-defined headers, employ bulleted or numbered lists, and use space to separate content.

- Does the user interface avoid overly bright or contrasting colors?

This is in conflict with advice that would be given for dealing with visual impairment. While it is a benefit to those with visual disabilities, such a stark contrast could be shocking or unsettling to someone with autism or another spectrum disorder. I also recommend limiting or removing elements that flash or otherwise draw attention in a jarring manner.

- Are text alternatives provided?

Having an audio or video file that goes with a text article can help users with dyslexia or dyscalculia better understand what is being presented. Many people with dyslexia use screen readers to help them digest and understand text passages, so the same techniques used to maximize the ability for screen reader use for those with visual impairments will also benefit those dealing with cognitive issues.

- Are there prompts and visual support for actions?

For many users with cognitive issues, it is not uncommon to lose track of what they are doing in the middle of a workflow. Being forced to remember a variety of steps to accomplish a task may be asking too much for many users. By providing prompts to highlight the previous and next steps, we can help them keep track of where they are and what they should be doing.

- Use breadcrumbs to identify where a user is in the page hierarchy

By implementing a breadcrumb option, you can validate that pages are appearing in the correct order, as well as determine that they can be navigated by following the breadcrumb trail as a series of links. This works in conjunction with testing the navigation elements on pages. Having a consistent design for both navigation elements and breadcrumbs can help users know where they are and what they are doing.

- Provide options for text resizing

When users can rescale the text they are viewing, it can make for a more comfortable reading experience. Having buttons that cannot be resized will cause pixelation when they are enlarged, in some cases making them illegible or indeterminable. By looking for elements that use the CSS box model and inline styles, those elements can be resized, keeping their actions recognizable to the viewer.

- Use the <ABBR> and <ACRONYM> HTML tags

It is common to see a variety of abbreviations or acronyms used across web pages, particularly in the software industry. These may not be understood by many readers, and not just those with cognitive disabilities. But if you use the <ABBR> and <ACRONYM> tags, users can hover over the letters and see what each abbreviation or acronym means in full text. You can make tests to search for all acronyms or abbreviations and determine whether accompanying tags and titles are provided.

- Reformat justified text

If a paragraph uses full justification, words will appear equally spread out along the line, instead of centered or aligned left or right. This can have the unintended consequence of making the material difficult to read for many people. A simple test that looks for and highlights paragraphs that are formatted with “<p align=”justify”> can help pinpoint where there may be difficulties.

Making our sites usable by the largest possible group of people is an important goal. Unfortunately, it is impossible to make every site completely accessible to those with cognitive impairments due to the range and varying severity of issues, but that should still be the aim.

Conclusion

The design decisions made early in the life cycle of products have the potential to make them excellent solutions to issues they face, or genuine nightmares to use. The farther along a product gets in its development, the more difficult it is to make modifications to its design.

Think of the applications that you would want to use, and think of yourself in the future, with the possibility that a significant disability (or disabilities) may be part of your everyday experience.

References

Wikipedia. “Accessibility”. https://en.wikipedia.org/wiki/Accessibility

Disabled World. “Defining Disability Diversity in Society”. https://www.disabled-world.com/disability/diversity.php

Census Bureau Reports. Nearly 1 in 5 People Have a Disability in the U.S. https://www.census.gov/newsroom/releases/archives/miscellaneous/cb12-134.html

World Health Organization. World Report on Disability. http://www.who.int/disabilities/world_report/2011/en/

North Carolina State University. The Principles of Universal Design. https://www.ncsu.edu/ncsu/design/cud/about_ud/udprinciplestext.htm

Sydik, Jeremy J. 2007. "Design Accessible Web Sites: Thirty-Six Keys to Creating Content for All Audiences”. Pragmatic Publishing

Bringing stakeholders together through modeling

Evan Masters

emasters@critical-logic.com

Abstract

Written specs are often confusing, ambiguous, incomplete, or simply gargantuan and overly complex. This leads directly to defects being built into the business systems that they attempt to describe. In this talk, I will describe a method that enables enterprises to deliver high quality business systems in a repeatable, predictable fashion using technology, standards, modeling principles, and automation to validate and verify system behavior.

We have all heard that a picture is worth a thousand words, so why do we use so many words to describe things instead of using pictures? In this talk, I will describe how the act of creating pictures in the form of models to visualize the intended behavior of a business system brings together stakeholders of every part of the Software Development Lifecycle (SDLC). These models augment the documentation created to convey business needs to the development team. Models can eliminate ambiguity, clarify confusion, and fill in the gaps from incomplete specs. They can also provide a simplified view into the complex, making what may seem massive more manageable.

I have personally been involved in more than two dozen development projects where models were created as part of the design phase, and my organization has been involved with over 100 such projects. The models were designed with the intent of using them for Model-Based Testing (MBT), but they served a useful purpose to nearly all stakeholders involved in the business system development process. In some projects, the organizations were able to do more with their existing staff. In others, software quality metrics were increased. Still others were able to begin implementing automated testing using these models as guides.

In all cases, visualizing the information in the form of models creates a common understanding of the desired behavior of the business system under development, reduces the defects introduced into the development process, and brings stakeholders together to take charge of quality at every stage of the design and development processes.

Biography

Evan Masters is passionate about seeing customers succeed. With over 10 years' experience in the software quality assurance industry, Evan has helped companies around the globe overcome challenges and implement innovative new technologies in support of business goals and objectives. Working with companies of all sizes, from startups to Fortune 5 companies, and thriving on his ability to listen to and understand customers' real needs, he has equipped and supported teams with the tools necessary to take their capabilities to the next level.

1 Introduction

To meet the ever-increasing demands of delivering high quality software products and solutions at a pace matching rapidly evolving business needs, Software Quality Assurance (SQA) professionals need to leverage every resource at their disposal. An effective strategy SQA professionals can employ to ensure the quality of the product or solution is to engage with stakeholders at every stage of the SDLC, not just the Testing stage. A tool that can be used to implement this strategy is a diagram of their understanding of the intended behavior of the business system. This helps to prevent any misalignment in the understanding of the intended behavior across the various stakeholder groups.

Creating (good) models and diagrams to augment text-based definitions of the desired system offers many benefits which I will discuss in depth throughout this work. Creating models and diagrams allows SQA professionals to become engaged much sooner in the SDLC than typically realized, putting quality front and center from the start of the process. These models and diagrams give the SQA professionals a layer of abstraction that can be used to convey an understanding of the system being developed in a way that stakeholders of any technical or business level of expertise can understand.

As organizations grow and SQA teams improve their capabilities, models and diagrams can evolve with them. SQA groups can leverage the models created early in the SDLC to design and generate their tests. This process is known as Model Based Testing, or MBT. SQA groups who automate their test execution can leverage these models even further, exposing their automation frameworks at the model level. This process gives stakeholders at every level the ability to create automation scripts, regardless of their technical expertise. I will describe this evolution from diagram and model creation, to Model Based Testing, to automatic script generation below.

2 Benefits of Creating (Good) Models and Diagrams

Before I dive into my description of how creating models and diagrams can bring stakeholders together to improve software quality, I will begin with a question: What is the end-goal of software development? The most limited and least prescriptive answer is “to deliver quality business systems.” This begs the follow up question: “does software development always achieve the goal it sets out to?” The obvious answer is no, it does not. Why is this? In practice, there are numerous reasons why development falls short of its goal; lack of time or budget, not having the required skill set, the limitations of technology, or most critically, a misunderstanding of the true business need, to name a few.

This misunderstanding can come from any number of sources. Sometimes the business itself does not know its real need or cannot articulate it sufficiently. Important aspects of the need are also often omitted or misconstrued when translated from business to technical terms. Other times, the development team is siloed and does not have the proper context to adequately grasp the business need and deliver accordingly. Most common, though, is that the business attempts to convey the need using traditional communication methods, such as a written specification or a requirements document, which fall short of providing enough detail to fully implement and test the system.

This is significant because recent studies[1] show that over 80% of software defects originate from the requirements elicitation and design stages of the SDLC. One of the primary reasons is that the written English language is inherently ambiguous, even to those with the highest degree of mastery. Many business stakeholders are adept writers, but this method of communication leaves much to be desired. One effective way to circumvent to downfalls of the written word is to augment specs and requirements with pictures that illustrate and help visualize the concept that is being conveyed.

As I mentioned in my abstract, a well-known saying goes: “A picture is worth a thousand words”. While there is some debate as to whether or not this adage holds true universally, I will accept it as a premise for the sake of this work. What kind of pictures, then, are useful when it comes to helping the SDLC achieve what it sets out to do? It turns out that there is no lack of useful visual aids; from simple ‘back of the napkin’ sketches, to abstractions brought to light on a whiteboard, to formal diagrams that have

defined standards and best practices. I will focus on the more formal types of pictures in this work; however, it is worth mentioning that less formal types can still be beneficial.

It is important to note that not all written specs are 'bad' specs. No one sets out to write a 'bad' spec; rather the spec feels accurate to the author who understands the need but fails to convey it to a reader is not already familiar with it. In other words, I could write a requirement or user story that is complete and clear from ***my*** perspective but can be interpreted very differently by someone else. I recently had a conversation with my boss where I said I was having a meeting with a customer *next Thursday* (we had the conversation on a Monday). He asked for clarification on the date. To me, next Thursday obviously meant the Thursday of the following week. He took it to mean the Thursday of the current week, being the next Thursday we would encounter. Even though ***I*** thought I was clear, there was ambiguity in my statement.

There are useful tactics and techniques that can be used to make written specs less ambiguous. Formal structured language, for example, can assign well-defined meanings to words that may other be interpreted multiple ways. In the 'next Thursday' example, I could have augmented the description of the day of the meeting with a numerical date. Instead of 'next Thursday', I could have said 'Thursday, July 15th, 2021'. There is only one way to interpret the date in this format.

It is worth mentioning that it is just as possible to create models or diagrams that are ambiguous or confusing as it is to author written specs that are ambiguous or confusing. One should not simply create models or diagrams for the sake of having them without having a strategy or plan in place on how they will be implemented. In order for models or diagrams to add value, they must be clear, well designed, and thought out. Fortunately, many types of diagrams have conventions, best practices, and even restrictions on what you can do in the diagram in order to help ensure the completed work does what it sets out to do.

So, what does it look like to apply the approach of creating models and diagrams (more on this distinction later) to the SDLC? Fortunately, we are able to create models and diagrams at every stage of the SDLC, leveraging the features and benefits of different types of models and diagrams that best suit the various stages. Some types of diagrams can evolve through stages of the SDLC, taking on more content or growing in detail as the project progresses. Shortly, I will discuss these features and benefits during the various stages of the SDLC, including types that can go through this evolution.

Before I discuss the features and benefits, it is important to state that one way to categorize models and diagrams is as such: Behavioral and Structural (also sometimes called dynamic and static respectively). The Object Model Group (OMG) maintains the standard for the Unified Modeling Language (UML) and describes the two categories as follows.

2.1 Behavioral Diagrams

"Behavioral Diagram: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams, and state machine diagrams."

2.1.1 Activity Diagrams

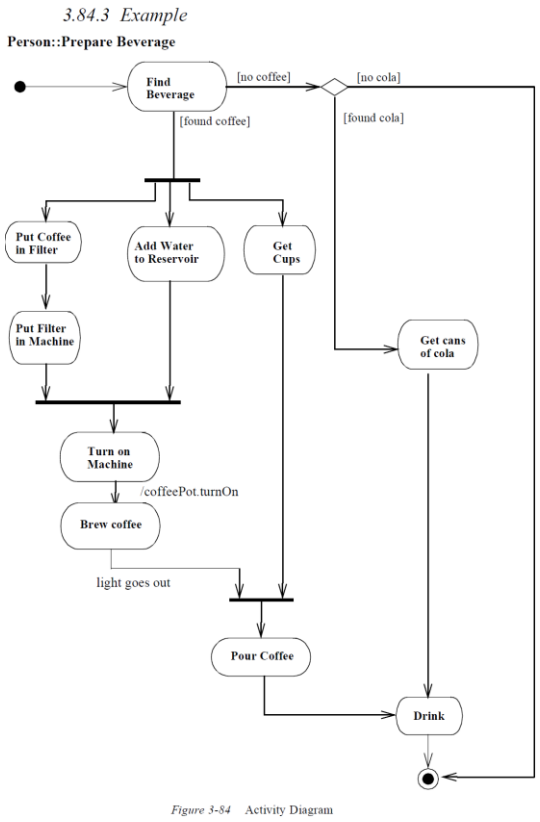


Figure 1: Example Activity Diagram[2]

An example of a Behavioral Diagram is an Activity Diagram. An Activity Diagram, as defined by the OMG, is *"…a special case of a state diagram in which all (or at least most) of the states are action or subactivity states and in which all (or at least most) of the transitions are triggered by completion of the actions or subactivities in the source states…The purpose of this diagram is to focus on flows driven by internal processing (as opposed to external events)."*

Activity diagrams show activities, nodes, splits and joins, and the 'flow' of the process through these elements. Critical Logic used Activity Diagrams at a major financial institution to help aide their IT department's transition from a Waterfall-oriented development methodology to an Agile development methodology. While the digital transformation had many aspects to it, introducing diagrams into the process streamlined communication between the team members. Business stakeholders were able to create a high-level version of the diagram. We considered this initial version the minimum viable product, or MVP, view of the business need. The MVP version was then handed off to the QA stakeholders who more fully fleshed out the diagram with exceptions and implementation-level details. The implementation version was then provided, along with the user stories and acceptance criteria, to the development team. The result was a common understanding of the business need at every step of the way, along with the following benefits:

1. A reduction in overall solution design and development time
2. Reduced time and cost of test design
3. A higher quality product delivered in each sprint

2.1.2 Use Case Diagrams

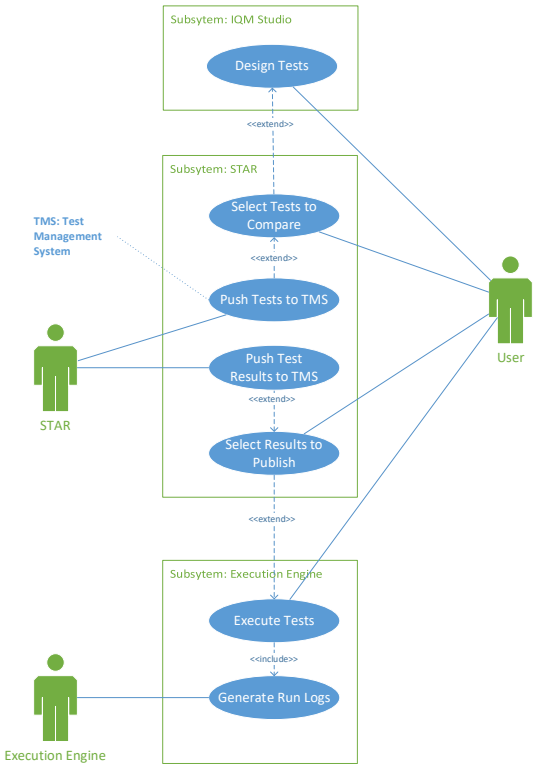


Figure 2: Example Use Case Diagram[3]

Another example of a Behavioral Diagram is a Use Case Diagram. A Use Case Diagram is *"…a graph of actors, a set of use cases, possibly some interfaces, and the relationships between these elements. The relationships are associations between the actors and the use cases, generalizations between the actors, and generalizations, extends, and includes among the use cases. The use cases may optionally be enclosed by a rectangle that represents the boundary of the containing system or classifier."*

Use Case diagrams, sometimes called Function Maps, show the different functions that make up a system as well as the actors that invoke those functions. Notice that each node, or Use Case, begins with a verb. Critical Logic utilizes Use Case Diagrams in its own internal development projects. Use Case Diagrams helped Critical Logic define the current state of the software product, IQM Studio. With a clear picture of their as-is system, Critical Logic was then able to define customer needs and identify functional gaps which could be put on their product roadmap to define their to-be system. In the theme of using diagrams to bring people together, stakeholders from all business areas (including Executives, Product Owners, software QA, and developers) were able to study this roadmap and see how the various functions of IQM Studio related to each other and understand the business value of implementing new functions.

2.2 Structural Diagrams

"Structural Diagram: emphasizes the static structure of the system using objects, attributes, operations and relationships. It includes class diagrams and composite structure diagrams.[4]"

Structural diagrams describe the static, unchanging elements of a system while Behavioral diagrams describe a system 'in action'. When used appropriately, both categories of diagrams serve a useful purpose for describing the business system and the two categories complement each other, meaning that both categories of diagrams can be used in parallel. It should also be noted that while there are currently

14 types of models and diagrams recognized by the OMG UML standard, many other types of diagrams exist that can be useful in their own right.

2.2.1 Class Diagrams

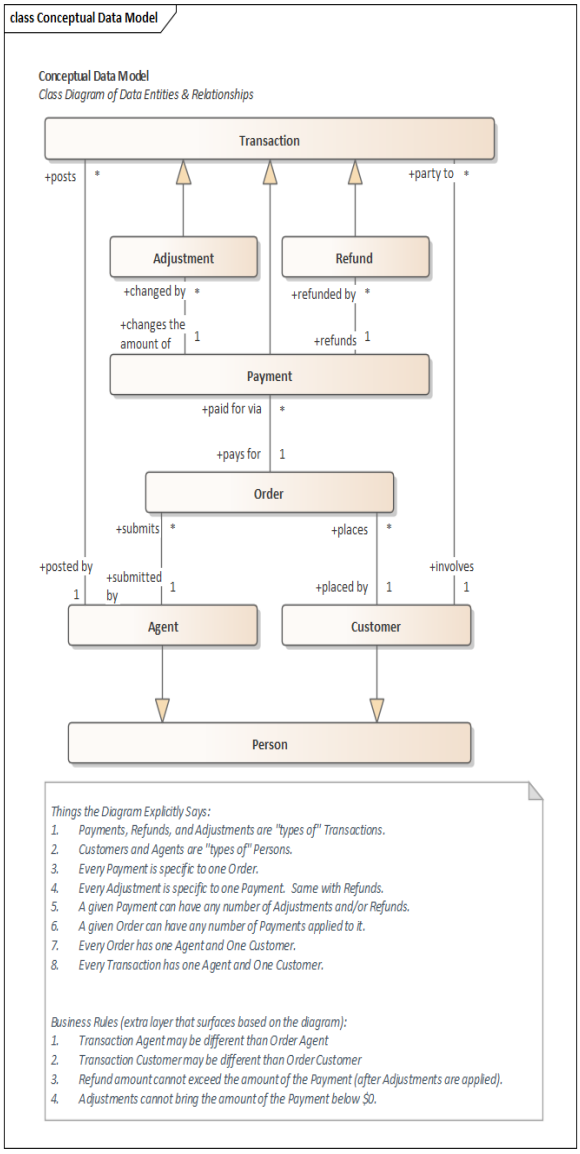


Figure 3: Example Class Diagram 1

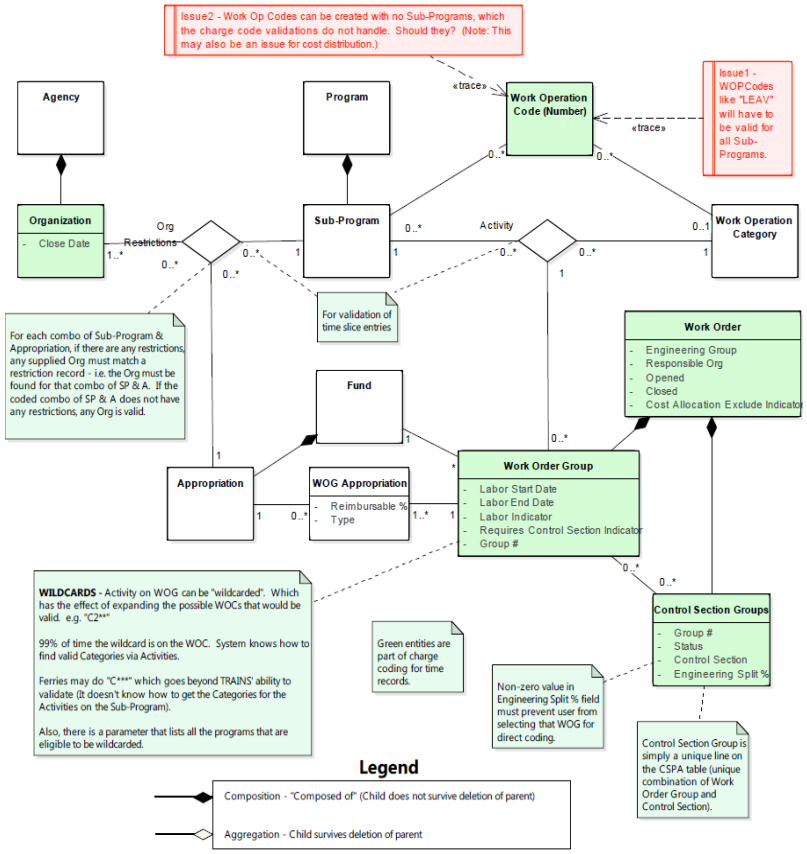


Figure 4: Example Class Diagram 2[5]

An example of a Structural Diagram is a Class Diagram. A Class Diagram is *"…a collection of static declarative model elements, such as classes, interfaces, and their relationships, connected as a graph to each other and to their contents."*

Class Diagrams show various levels of detail of the system they represent. The first example above shows only the classes and their relationships, while the second example includes attributes about the classes. Class diagrams are often thought of as technical diagrams used only by Object-Orient programmers. Critical Logic has used Class Diagrams as a powerful tool for bridging the gap between business and technical stakeholders through an abstraction layer. Class Diagrams also provide the vocabulary for rule and requirement writing, again allowing different stakeholders to communicate using the same language.

2.2.2 Component Diagrams

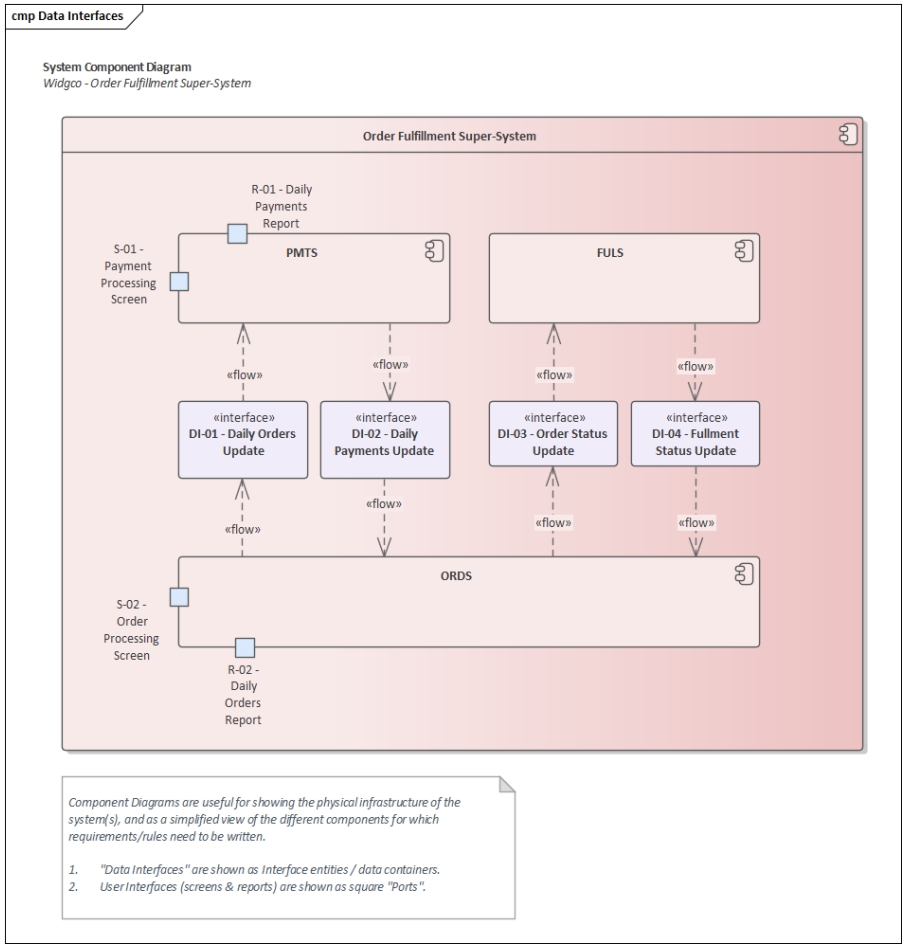


Figure 5: Example Component Diagram

Another example of a Structural Diagram is a Component Diagram. A Component Diagram is *"…**a** graph of components connected by dependency relationships. Components may also be connected to components by physical containment representing composition relationships."*

Like Class diagrams, Component Diagrams are often thought of as technical diagrams used only by Object-Orient programmers. They also serve as a powerful tool for bridging the gap between business and technical stakeholders through an abstraction layer. Critical Logic has developed Component Diagrams on customer projects to bring attention to the various interfaces of a system or systems that are part of the implementation solution. Component Diagrams can be effective for developing a holistic view of the solution as well as everything the solution touches.

2.3 Uses of diagrams during the SDLC

There are no hard and fast rules for which models and diagrams should be created during the various stages of the SDLC, though some do tend to lend themselves to be most effective at certain times. One approach to selecting a type of diagram is to create them that are strategic in nature early on in the process and, as the process advances, transition to models and diagrams that are more functional in nature. For example, while in the planning and analysis phases, creating diagrams regarding organization structure (Organizational Charts), As-is and To-be processes (Business Process Models), business activities (Activity Diagrams), or associations (Entity Relationship Diagrams) could be quite useful. As the project advances to the design, development, testing, and implementation phases, creating models and

diagrams regarding specific functions (Use Case Diagrams, sometimes referred to as Function Maps), system architecture (Database Diagrams, Component Diagrams), intended system behavior (Cause-Effect Models, Activity Diagrams), or system transitions (State Machine Diagrams) will greatly augment the text-based definitions of the system.

Notice that Activity Diagrams are effective to create in both the early stages of the SDLC as well as the latter. As mentioned previously, some types of diagrams lend themselves well to evolving through the SDLC. Cause-Effect Models are another type that can evolve. Models and diagrams that can evolve throughout the process are especially valuable because they continue to be useful as the process advances from one group of stakeholders to the next, adding a level of continuity that could otherwise be lost.

There are benefits to creating diagrams in both pre-development and post-development stages of the SDLC. When creating models early on, some of the benefits include:

- Developing a clear understanding of the need to be met
- Accelerating the design stage
- Allowing questions to be brought to light and discussed by the appropriate stakeholders
- Unveiling ambiguity that may exist in the business requirements
- Reducing maintenance burden in the Maintenance and Operation stages

Creating models and diagrams helps stakeholders visualize the need in a way that can be hard to do in pure text-based descriptions. Similarly, when designing the 'what' part of the way the need will be met (in a solution-independent fashion), creating models and diagrams reinforce that what will be built ***should*** be built. For example, a Use Case Diagram that shows a complete, non-overlapping list of all functions the new system will contain will much more effectively and efficiently be shared among the vested stakeholders than a text-based document on its own. This in turn will help the design come together more efficiently with less back-and-forth between stakeholders. This greatly reduces the length of the design phase. Models and diagrams hold a significant advantage when it comes to bringing ambiguity regarding the intended behavior of a system to light. The ambiguity can then be discussed and driven to resolution in the early stages of the SDLC, saving cost and rework from occurring later on.

Creating and maintaining models and diagrams in the later stages of the SDLC also provides number benefits:

- Provides a clear sense of how the solution will be implemented
- Unveils ambiguity that was not discovered during initial design phase
- Clarifies confusion
- Fills in the gaps in the spec
- Makes the massive manageable
- Creates a simplified view of the complex
- Can be fleshed out to contain very deep level of detail (e.g., equivalence classes, boundary values, combinations)
- Some types allow for tests to be derived from the diagram – others can even automatically generate tests from the diagrams

When the business system being developed has a graphical user interface, or GUI, models and diagrams are effective to convey what the GUI should look like. For example, imagine how much effort it would take to describe what a website's homepage should look like without using models or diagrams (such as mockups or wireframes). Even when the system does not have a GUI or there are GUI-independent components to the system, models and diagrams are still useful to describe how these components should work. Using a Database diagram to represent the architecture of a backend database is significantly more efficient than an attempt to describe the architecture using only words.

As with creating models and diagrams in the early stages of the SDLC, creating them during the later stages helps to bring to light questions, ambiguities, and gaps in the specifications, again saving cost and

preventing rework later on. This also is an effective way to eliminate defects from being introduced into the system due to unclear specs.

Models and diagrams also serve a helpful purpose when it comes to defining scope. Certain types of diagrams, Use Case Diagrams for example, can be used to fully define the functions of a system, making what seems like a massive, complex interaction of disparate things seem manageable and understandable.

When models and diagrams are leveraged to capture low levels of detail, they serve multiple purposes. For example, when Cause Effect Diagrams are used to define the intended behavior of a system, they can take on increasing levels of detail as a project moves from design to development to testing and, finally, the implementation stage. A Cause Effect Diagram can start out defining the 'go-right' behaviors of a system early on and then have more details, such as exceptions or alternate paths added as they are defined. Cause Effect Diagrams are also excellent for defining boundary value conditions, equivalence classes, and logical combinations. Certain types of Cause Effect Diagrams can even automatically generate high coverage test cases from the information in the diagram, an effective type of Model Based Testing (MBT).

A major benefit of MBT is that the models themselves are multi-use artifacts. Many other artifacts developed during the SDLC are single-use. Functional specifications, for example, serve the sole purpose of specifying how the solution will be implemented. While they can serve as a guide to developing other artifacts, it still requires an additional independent step to generate the additional artifacts.

Models used in MBT, however, are different. They serve the valuable initial purpose in bringing stakeholders of various levels together to give them a communication conduit that develops a common understanding of the business need and solution to be implemented. These models can go on to serve an additional purpose when they are used to create tests from the information defined in the models. Some commercially available tools can even *automatically generate* tests from the model. This allows the model to continue to add value throughout the stages of the SDLC, including into the maintenance and operation phases. The next section will expand on the benefit recognized by implementing MBT.

3 Benefits of Moving to Model Based Testing

Model Based Testing has been increasing in popularity, but it is far from a new concept. In 2008 a Department of Defense contractor, General Dynamics, was challenged by the United States Navy to reduce testing costs while at the same time increase the quality of their mission- and life-critical systems, which was no small feat[6]. A year later, they were introduced to an MBT solution that they implemented for their Navy project. In addition to saving General Dynamics $3 Million on that project alone, Ron Townsen, the Sr. Lead Engineer at General Dynamics had this to say:

"[DTT, the modeling tool], along with the higher quality of model-based testing, gave us the approach we have been looking for...Especially in areas of complex design and safety critical needs"

General Dynamics is not alone in their implementation of MBT. The Model-Based Testing User Survey has been administered and published by a number of individuals since at least 2011. The most recent survey was administered and published in 2019 by two members of the German Testing Board, Anne Kramer and Bruno Legeard. One question on the survey asked respondents about their expectations for implementing MBT at their organization, to which over 60% of respondents answered, "We wish to improve the communication between stakeholders." The survey goes on to ask, "From your current experience, does MBT fulfill those expectations?". A staggering 75% of respondents answered "Yes" (46%) or "Partly" (29%) to this question.[7]

Studying additional questions from the survey reinforces the idea that models and diagrams can be used at any stage of the SDLC, requirements elicitation for example, to great benefit. These questions also demonstrate how models and diagrams evolve from when they are built as they progress from stage to stage.

Another excellent example of this is Sun Microsystems, which turned to MBT to help with their unique challenges after being acquired by Oracle in 2010[8]. At the time, Sun's 'Configurator' was a system designed to give their sales force a current view of their product offering, ensuring that the most current and compatible options were offered. Given the rapidly changing state of the complex hardware and software landscape, the Configurator's nearly 60,000 business rules were constantly in a state of flux. In a given month, an average of 25% of the rules would undergo some form of change. By representing these rules in Cause Effect Diagrams, then automatically generating tests from the diagrams, several significant benefits were realized.

First, the rate of release for the configurator doubled, increasing from once every other month to once a month. This means that the Configurator was always returning the most up to date offering of Sun's product line. It also ensured that quotes are accurate and timely.

Second, several consecutive zero-defect releases went into production. This was something that Sun had not been able to accomplish prior to implementing MBT with Cause Effect Diagrams. These zero-defect releases give their sales team confidence that the information they are giving their customers is correct and reliable.

Third, the amount of testing resources to accomplish these achievements actually ***decreased by a third***. By freeing up testing resources from manual testing, Sun's SQA teams are able to focus on value-added QA activities. Sun also reduced their head count of testing resources, allowing these resources to be moved to other areas such as to the dev team.

Salesforce is another household name that has benefited significantly from implementing MBT[9]. Straining from rapid growth due to their successful platform and services, Salesforce.com had no consistent process for capturing requirements during the elicitation stage. This led, unsurprisingly, to missed or incorrectly implemented requirements causing defects in production. Given Salesforce's reputation as a world leader in Software as a Service, this was not sustainable. As mentioned above, the requirement authors at Salesforce.com set out to create complete, unambiguous, and implementable requirements. In their case, and many cases just like this, it was a limitation of resources available that sabotaged their ability to achieve their goal.

Within weeks of implementing Model Based Testing and focusing on visualizing requirements for clarity and completeness, Russ Nelson, the Vice President of Applications Development had this to say:

"[The MBT implementor] reduced our requirements elicitation investment by 50% while dramatically increasing the quality of the software we deliver…[and] they allow us to leverage our existing staff 4:1."

What does the process of creating Cause Effect Models which automatically generate tests look like in practice then? Let's walk through it.

3.1 Model Based Testing in Practice

The first step in a disciplined and effective approach to modeling requirements is to perform an initial analysis of the requirement artifacts that have been developed so far. To illustrate this, I will describe another real-world application. Critical Logic engaged in a project in 2015 – 2016 to assist in the modernization of a Veteran's Affairs (VA) system that would centralize and make nationally available medical records of US Armed Forces Veterans, specifically their medications and adverse reactions to medications. By the time Critical Logic engaged in the project, all of the business and functional requirements had been defined by the VA business sponsor and turned over to the vendor for implementation.

After winning the contract and taking delivery of the requirements, the vendor quickly realized that the 'complete' requirements were anything but. This was a good realization, but they were left without a clear path forward to develop the solution the VA would accept. The requirements were simply not complete and too ambiguous for development and implementation. Due to the nature of the contract, requesting new requirements was also not an option. They needed an approach that would allow them to leverage

the work the VA had done AND result in a complete requirement set, which is when Critical Logic became involved.

Critical Logic's first task was to represent the requirements in Use Case Diagrams. Doing so demonstrated the areas that the VA requirement authors had failed to define. The Use Case Diagrams proved effective at showing exactly where additional requirement elicitation or clarification was required. This approach allowed the vendor to go back to the VA in a manner that showed they were leveraging all of the work the VA had undertaken in authoring the requirements, while asking only targeted questions to fill the requirement gaps and clarify ambiguities.

Once a *truly* complete set of requirements had been defined, Critical Logic began representing the requirements in Cause Effect Diagrams. Cause Effects Diagrams created at this stage built upon the work done in the Use Case Diagrams and included functional requirement-level information that unearthed additional requirement gaps and ambiguities. The image below is an example of a Cause Effect Diagram from the project that unearthed functionality that was not defined well enough to be implemented by the development team.

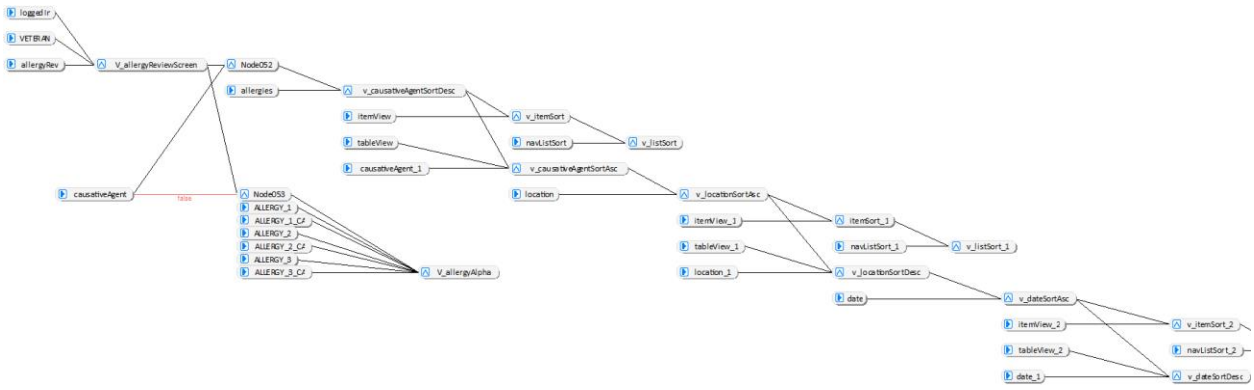
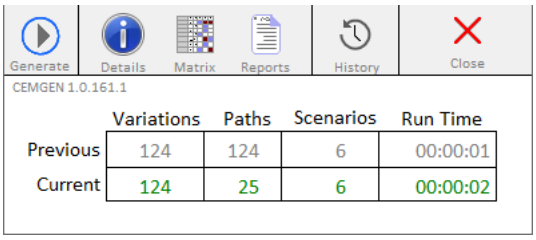


Figure 6: Real-world Cause Effect Diagram

In general, each function defined in the Use Case Diagrams corresponded to a set of functional Cause Effect Diagrams. Once a Cause Effect Diagram was created and taken to a point where it was ready for review, Critical Logic would review the diagram with the VA and Vendor subject matter experts (SMEs) in ambiguity review sessions, bringing to light any questions that arose during the diagram creation process. When questions could not be answer in real-time, Critical Logic would track the ambiguity and ensure that it was driven to resolution.

Once the diagrams had resolved all of the requirement gaps and ambiguities, a test generation algorithm was invoked that took the information in the Cause Effect Diagram as input and automatically generated tests from the model.



	Variations	Paths	Scenarios	Run Time
Previous	124	124	6	00:00:01
Current	124	25	6	00:00:02

Figure 7: The test generation algorithm created 6 scenarios from the diagram

Ideally at this point, the tests would be reviewed by the SMEs for sign off. Because the tests had been generated from the model, and because the model had been reviewed by the VA and Vendor SMEs, the test review process was incredibly efficient, and signoff was given with minimal updates needed.

The benefits of using Use Case and Cause Effect Diagrams to augment and enhance the VA-authored requirements were apparent in the first sprint retrospective where the business was able to see the MVP. Sprint after sprint, as the VA and vendor teams embraced the CEM process and the ambiguity and test review sessions, efficiency increased, and the development team consistently delivered nearly defect-free product that truly represented the business needs. Ultimately, the product was accepted, and the implementation was successful.

In each of these cases, we saw organizations with different challenges and goals; all of which were realized with the assistance of MBT. General Dynamics was able to improve the quality of their mission- and life-critical systems while at the same time reducing their testing costs. Sun Microsystems were able to keep up with the frenetic rate of change required for them to maintain their market-leading status. Salesforce.com saw that they could improve and standardize their requirements elicitation process while at the same time allowing their staff to be more efficient with their time. The VA implementation vendor took the incomplete, ambiguous requirements authored by their customer and delivered a truly complete set of requirements and a product that fully represented these requirements.

In each example, creating models provided the project artifacts that allowed stakeholders to come together and enhance the quality of the system in development. With General Dynamics and the VA vendor, the models provided a graphical representation that was easily consumable by the business and dev teams. By discussing questions in the context of the graphical representation of the requirements, the gap between the business and development stakeholders was bridged. With Salesforce.com, CEMs were used to enhance the elicitation process, putting the end users and business analysts on the same page. At Sun, CEMs brought stakeholders together to allow their business and development teams to keep up with the required rapid pace of change that Sun needed to maintain their market leader status.

4 Moving from Model Based Testing to Automated Testing

Once an organization has progressed to the point where they are realizing the benefits of Model Based Testing, there is an additional level that can be achieved to further promote unity between all stakeholders. Before I get into the details of this final step, however, let's discuss the distinction between a model and a diagram.

A model is an abstract representation of something, a system for example, where every part of that system is defined exactly once. A diagram is an instance of the model. Each element that makes up that system is referenced when that element is needed to define the system in a diagram. Each time an element is used, it is a *reference* to the element. This eliminates the possibility of having the same element described in more than one way and ensures that each element referenced is exactly the same each time it is used.

A diagram can be thought of as a specific view of the model. Different elements from the model can be assembled in a way to create the view you are interested in, and the same elements from the model can be used to create any number of different diagrams.

This distinction between models and diagrams is important when talking about automated testing because there is an incredibly effective way to leverage models, MBT, and automated testing frameworks that takes advantage of the concept of a model. This concept was put into practice at General Dynamics, as mentioned above. Specifically, the teams at General Dynamics created Cause Effect Models to represent the two major components of their integration project. These two systems combined had over 2,500 requirements to define their intended behaviors. Even the most skilled analyst could never ensure that this was a complete, unambiguous, non-contradictory list of statements using only formal structured text. By representing these requirements in a disciplined, structured fashion using Cause Effect Models, the teams were able to realize all of the benefits mentioned above. They raised questions via ambiguity review sessions in a regular cadence and were able to get clarification and improve the requirements before they were approved and sent to the development team. Eventually, the development team started

sending a representative to participate in the ambiguity review sessions to get a firsthand account of the clarification process. In this instance, modeling helped bring stakeholders together in a situation where they otherwise would have remained siloed.

Once the Cause Effect Models were created, the team was able to automatically generate a full coverage set of tests from the models. Because the Cause Effect Models were created in the design stage, and the tests were generated from the models, the tests were developed much sooner in the SDLC than typical. This meant that the tests were able to be sent to development team along with the requirements. The benefit of this was that the development team knew exactly what tests would be executed in order to validate the system behavior, giving them an additional design artifact to base their development on.

The General Dynamics team did not stop there, though. They then created an automation framework that was exposed at the model level. Doing so allowed them to build an automation test script library that correlated perfectly to the tests generated from the models by treating the automation framework like a model - each object in the system under test was defined once and then called each time it was a part of a test. These objects and their applicable actions were mapped to the steps defined in the models. After these objects and actions were mapped, or scripted as it is often referred to as, the tests automatically generated from the models also automatically generated the automation scripts.

To illustrate this process, here is a generic example that shows an example of a Cause Effect Model that is used to generate tests, and the corresponding scripting that allows the model to also generate automation scripts. This generic example represents a Library Information System, or LIS.

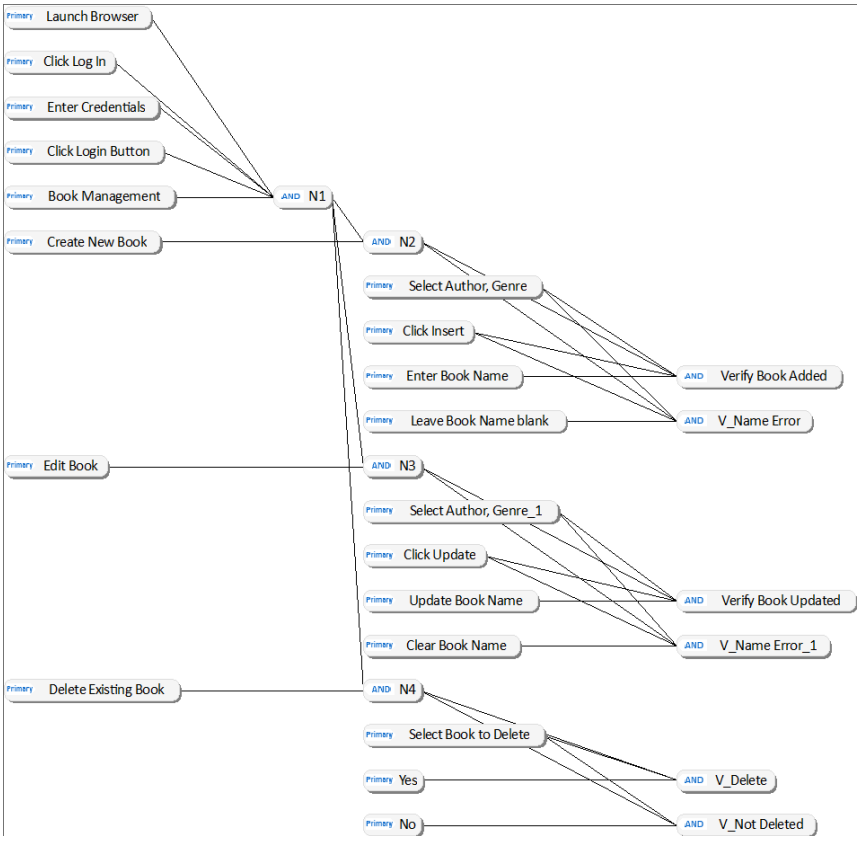


Figure 8: Cause Effect Model representing the LIS requirements

Here's what the LIS looks like:

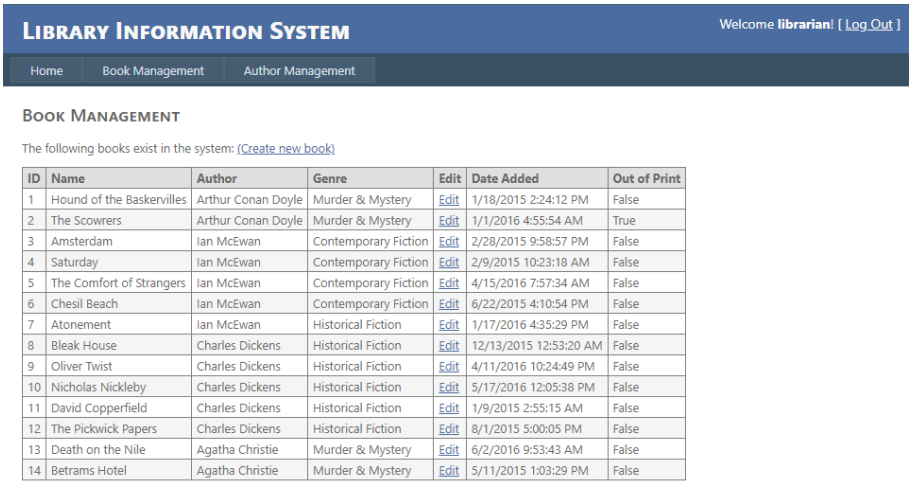


Figure 9: The LIS sample application[10]

Once a system such as the LIS is in production, an automation engineer can develop an automated testing framework of the system, mapping out all of the objects and assigning the valid actions that can be taken against those objects. For example, the user is able to click the 'Edit' link for a book in the LIS table. In this case, the 'Edit' link is the object and 'click' is the action. (Specifically, left-mouse button clicking is the action.) If we look at a subset of the CEM diagram, we see that the model represents the 'Create New Book' function:

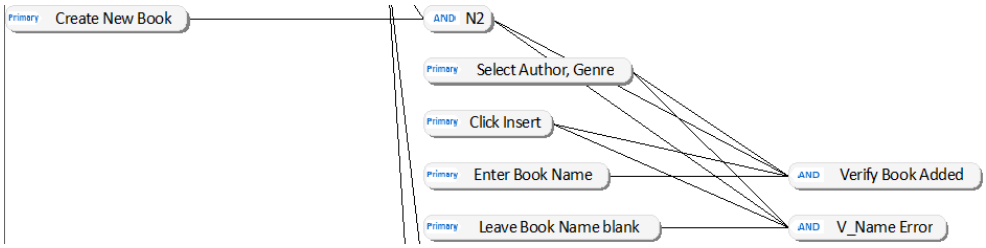


Figure 10: Subset of the LIS CEM

In the LIS itself, this is what the 'Create New Book' function looks like:

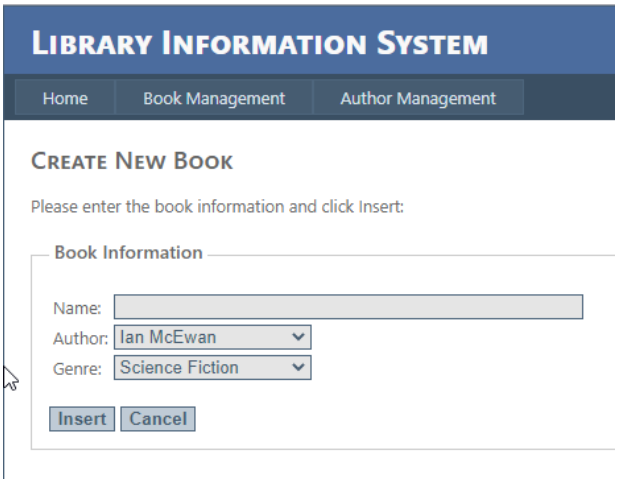


Figure 11: Create New Book function of LIS

The objects we are interested in for the sake of creating automated tests are Name (text field), Author (drop down box), Genre (drop down box), Insert (button), and Cancel (button). Each of these objects have different actions that can be taken against them. Take the “Name” field for example. Some of the testing-related actions we could take against the “Name” field are: Input text, Input text without clearing existing text, Verify it is enabled, and Verify it contains a certain value. Critical Logic’s commercially available tool, IQM Studio allows access to the automation framework with a simple drop-down selector tool that is human readable. In other words, non-technical stakeholders who create or interact with CEMs are able to ‘script’ the models by selecting these objects and actions. This is what it looks like in the tool:

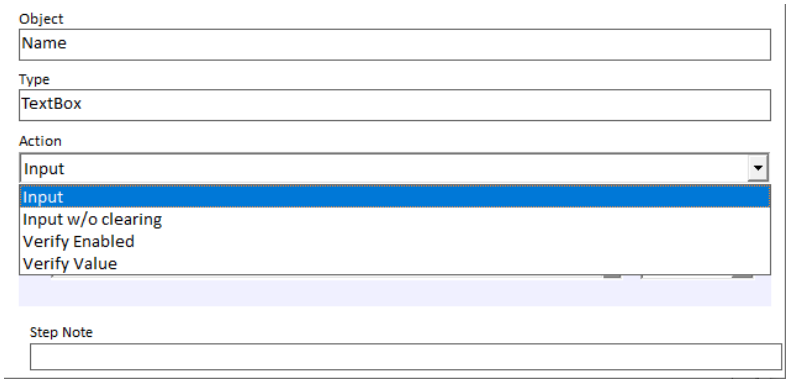


Figure 12: IQM Studio scripting and object and actions

All the modeler has to know is which object is being represented in the model and then choose which action is being taken against that object. Behind the scenes, IQM Studio is linking the scripted steps (the object/action pairs) to the code in the automated testing framework developed by the automation engineer. This splits the automation tester role into two: the automation engineer and the scripter. The automation engineer is responsible for developing and maintaining the automation framework code; that is, the code needed for the automation tool to take actions against the defined objects. The scripter is responsible for associating the proper objects and actions to the correct steps in the CEM. By doing so, IQM Studio’s test generation algorithm automatically generates tests ***and*** automated test scripts at the same time!

Step	**Action/Expected Result**
Start	
1	Node [Book Management V5.Launch Browser(T)] - Launch Browser and navigate to http://www.libraryinformationsystem.org
1.1	Verify the textbox 'Page Title' has the value 'Library Information System'
2	Node [Book Management V5.Click Log In(T)] - Click Log In
2.1	Click the link 'Log In/Out'
3	Node [Book Management V5.Enter Credentials(T)] - Enter "librarian" in the 'Username:' field Enter "librarian" in the 'Password:' field
3.1	Enter 'librarian' into textbox 'Username'
3.2	Enter 'librarian' into textbox 'Password'
4	Node [Book Management V5.Click Login Button(T)] - Click the "Log In" button.
4.1	Click the 'Log In' button
5	Node [Book Management V5.Book Management(T)] - Click the "Book Management" tab.
5.1	Click the 'Book Management' button
6	Node [Book Management V5.Create New Book(T)] - Click the "Create New Book" link.
6.1	Click the link 'Create new book'
7	Node [Book Management V5.Enter Book Name(T)] - Enter "Test Book: Now in color!*" in the 'Name' field.

Figure 13: Example model generated test

```
function Test(params)
{
   test:
   {
      //TMX Variables
      //tmxTestName = "BookManagementV5_Test_001"
      //tmxUser = "cli"

//--------------------------------------------------------------------------------------------------------------------------------------------------------
   stepSetup(1, "node", "Node [Book Management V5.Launch Browser(T)] - Launch Browser and navigate to http://www.libraryinformationsystem.org");
//--------------------------------------------------------------------------------------------------------------------------------------------------------
      tmxNode = "Book Management V5.Launch Browser(T)";
      log();     //continue on error


//--------------------------------------------------------------------------------------------------------------------------------------------------------
   stepSetup(1.1, "Verify Text", "Verify the textbox 'Page Title' has the value 'Library Information System'");
//--------------------------------------------------------------------------------------------------------------------------------------------------------
      var sTxt = SeS('Library_Information_System').GetText();
      var arg = "Library Information System";
      if (sTxt != arg)
      {
         rc = false;
         tmxLogComment = "Expected '" +arg+ "', Actual '" +sTxt+ "'"
      }


      if (log() != true)
      {break test;}  //stop on error

//--------------------------------------------------------------------------------------------------------------------------------------------------------
   stepSetup(2, "node", "Node [Book Management V5.Click Log In(T)] - Click Log In");
//--------------------------------------------------------------------------------------------------------------------------------------------------------
      tmxNode = "Book Management V5.Click Log In(T)";
      log();     //continue on error

//--------------------------------------------------------------------------------------------------------------------------------------------------------
   stepSetup(2.1, "Click", "Click the link 'Log In/Out'");
//--------------------------------------------------------------------------------------------------------------------------------------------------------
      SeS('Log_In')._DoClick();
```

Figure 14: Example automation script corresponding to manual test

What this means is that the testers (who are also the modelers and scripters) don't need to know how code and the automation engineer doesn't have to design, author, or maintain tests – only the framework itself!

5 Conclusion

The simple act of drawing a picture can convey so much meaning, information, and context. Creating models and diagrams to augment text-based definitions of a business system's intended behavior adds value at every stage of the SDLC. One of the primary ways models and diagrams do this is by creating a conduit of communication between stakeholder groups who do not always speak the same language. The models and diagrams allow stakeholders of varying degrees of business and technical expertise to have something to point to in order to help them convey concepts and ideas.

More mature organizations can leverage certain types of models and diagrams that allow them to implement Model Based Testing. Because models and diagrams can be created as early as the design phase, MBT can provide the benefit of developing tests much earlier in the SDLC than is typically realized. This provides an additional, valuable design artifact that can be supplied to the development team to provide an extra level of description into the intended behavior of the system being developed.

Finally, organizations can further leverage the models to build an automation framework that has the primary characteristics of a model. This allows teams to access the automation framework at the model level and generate automated testing scripts that can be executed once the business system has been deployed. This means that all people, stakeholders at *any* level, can take control of the quality of their organization's business systems, and the models are a key artifact in enabling them to be able to do so.

6 Selected Glossary of Terms (as defined by the OMG)

1. **State Machine:** The State Machine package is a subpackage of the Behavioral Elements package. It specifies a set of concepts that can be used for modeling discrete behavior through finite state-transition systems…State machines can be used to specify behavior of various elements that are being modeled. For example, they can be used to model the behavior of individual entities (such as, class instances) or to define the interactions (such as, collaborations) between entities. In addition, the state machine formalism provides the semantic foundation for activity graphs. This means that activity graphs are simply a special form of state machines.
2. **Activity Diagram:** A special case of a state diagram in which all (or at least most) of the states are action or subactivity states and in which all (or at least most) of the transitions are triggered by completion of the actions or subactivities in the source states…The purpose of this diagram is to focus on flows driven by internal processing (as opposed to external events). In the context of this paper, workflow diagram or swim lane diagram are not regarded as Activity Diagram, and they are out of scope of this paper.
3. **Activity Graph:** The Activity Graphs package defines an extended view of the State Machine package. State machines and activity graphs are both essentially state transition systems, and share many metamodel elements. An activity graph is a special case of a state machine that is used to model processes involving one or more classifiers. Its primary focus is on the sequence and conditions for the actions that are taken, rather than on which classifiers perform those actions. Most of the states in such a graph are action states that represent atomic actions; that is, states that invoke actions and then wait for their completion. Transitions into action states are triggered by events, which can be:
    a. the completion of a previous action state (completion events),
    b. the availability of an object in a certain state
    c. the occurrence of a signal, or
    d. the satisfaction of some condition.

References

1. https://www.academia.edu/2182202/The_pattern_of_software_defects_spanning_across_size_complexity
2. https://www.omg.org/spec/UML/1.4/PDF
3. Copyright, Critical Logic, 2021
4. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/behavior-vs-structural-diagram/
5. Copyright, Critical Logic, 2021
6. https://www.critical-logic.com/assets/General-Dynamics-Case-Study-V5.pdf
7. https://www.cftl.fr/wp-content/uploads/2020/02/2019-MBT-User-Survey-Results.pdf
8. https://www.critical-logic.com/assets/Critical-Logic-SunMicroSystems-CaseStudy.pdf
9. https://www.critical-logic.com/assets/Critical-Logic-SFDC-Case-Study.pdf
10. http://www.libraryinformationsystem.org/Books.aspx

Managing Accessibility in Software Systems

Jack McDowell, MA
Office of the State CIO
State of Oregon
jack.mcdowell@oregon.gov

Ying Ki Kwong, PhD, PMP
Office of the State CIO
State of Oregon
ying.k.kwong@oregon.gov

Abstract

The goal of accessibility in software systems is to produce software that is accessible to users independent of limitations that they may have. Although accessibility is frequently discussed as a "feel good" topic, along with usability and other nice to haves, the authors believe that accessibility should be understood to be a central component for software quality, due to the tangible and intangible risks associated with inaccessible software. In order to achieve the goals of inclusive software, accessibility must take on a broader role and move beyond testing against predetermined standards and instead be built into the Software Development Lifecycle as a crucial component of a software project. By doing so, we can expand the margins of our community of meaning to which a software speaks.

This paper seeks to explore the following topics:

- What is accessibility, and why it's important?
- How do we expand our understanding of accessibility in the context of Diversity, Equity, and Inclusion?
- How should accessibility be managed in this expanded "community of meaning" for the quality of information technology?

Biography

Jack McDowell works with the State of Oregon's Statewide QA Program. The Statewide QA Program provides Quality Assurance services for Oregon's Major IT Projects, and Quality Assurance consultation to Oregon State Agencies. Before this, he was a web developer and the chief editor of a community newspaper in Arlington, Virginia. Originally from Buenos Aires, Argentina where he lived before attending college in the US. He holds a Master's degree in political science from the University of Oregon and a certification in ITIL.

Ying Ki Kwong is the E-Government Program Manager with the State of Oregon. Prior he was the IT Investment Oversight Coordinator in the same office and was Project Office Manager of the Medicaid Management Information System Project in the Oregon Department of Human Services. In the private sector, Dr. Kwong was CEO of a Hong Kong-based internet B2B portal and a program manager in the Video & Networking Division of Tektronix. In these roles, he has managed software based systems/applications, products, and business process improvements. He received the doctorate from the School of Applied & Engineering Physics at Cornell University and was adjunct faculty in the School of Business Administration at Portland State University. He holds the PMP certification since 2003.

1 Introduction

The goal of accessibility in software systems is to produce software that is accessible to users independent of limitations that they may have. Although accessibility is frequently discussed as a “feel good” topic, along with usability and other nice to haves, the authors believe that accessibility should be understood to be a central component for software quality, due to the tangible and intangible risks associated with inaccessible software.

Accessibility has traditionally been focused on physical disabilities, and therefore building in accessibility usually means complying with WCAG 2.0 AA, section 508, or similar standards. Equating accessibility with standards compliance creates two important issues.

First, while content may be technically accessible, standards rarely measure the ease of use and understanding of the content which they are evaluating. Standards, in order to be broadly applicable and non-prescriptive in their implementation, set guidelines which can be achieved through different means, for example, whether an alt tag is descriptive and representative is up to the content creator and the auditor.

Second, by thinking of accessibility in terms of diversity, equity, and inclusion, we can and should expand the scope of accessibility beyond physical disabilities to also include cultural, linguistic, educational and related socio-economic factors in our understanding of accessibility.

In order to achieve the goals of inclusive software, accessibility must take on a broader role and move beyond testing against predetermined standards and instead be built into the Software Development Lifecycle as a crucial component of a software project. By doing so, we can expand the margins of our community of meaning to which a software speaks.

This paper seeks to explore the following topics:

- What is accessibility, and why it’s important?
- How do we expand our understanding of accessibility in the context of Diversity, Equity, and Inclusion?
- How should accessibility be managed in this expanded “community of meaning” for the quality of information technology?

2 What is Accessibility in IT Systems?

In order to address why accessibility is important, and how a broader understanding of accessibility can improve software quality, we must first understand the typical definition of accessibility. Accessibility as is normally understood in IT systems means that software products, web pages, etc. can be used by people with a variety of disabilities independently, that is, without a chaperone or employee assistance (e.g. for accessible service please call this number).[1]

2.1 The role of standards.

While accessibility itself is a broadly defined topic, it is typically understood through standards, mainly the Web Content Accessibility Guidelines (hereafter WCAG) and Section 508 of the Rehabilitation Act of 1973 (hereafter Section 508). WCAG is an international standard published by the Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C). Section 508 is a U.S. federal standard that

[1] For additional information see https://digital.gov/resources/introduction-accessibility/

mainly applies to U.S. Federal agencies, but may also apply to entities receiving federal funding, such as states or contractors.

These standards are used in order to determine whether a system is accessible. For example, there is a specific section of WCAG 2.0 which addresses which attributes a non-text image should have in order to be considered "accessible", such as including attributes which provide a text content equivalent so that screen readers can interpret the meaning of the page[2]. The WCAG standard includes guidelines that cover the gamut of web elements and situations such as video content, text contrast, headings, etc., the idea being that by following these guidelines, websites and applications will be accessible.

Although not always optimal, accessibility features to support physical disabilities can be retrofitted or integrated into an existing software system to accommodate end-user needs. In practice, developers use open source or commercial software components / scripts to enhance different parts of a code base. This is usually done by modifying templates or available services, in order to achieve compliance with applicable technical standards, e.g. WCAG 2.0 or HTML5. These approaches can be quite effective as aspects of assistive technologies (AT), because modern browsers have built-in support to interpret and render accessibility features, such as ARIA label (Accessible Rich Internet Applications labels for labeling a web element) and Alt text (alternative text for describing an image). Incorporating these AT elements in a template can go a long way toward improving accessibility but, of course, *only to the extent anticipated and enabled by the underlying standards*.

To assure the quality of accessibility in this context, current best practices may call for different types of "accessibility testing." First, a software system can be subjected to testing using automated tools, in order to identify non-compliance with one or more technical standards. Second, users with actual accessibility needs may act as testers during User Acceptance Testing (UAT). As an example of this type of accessibility testing, the state of Oregon may involve the Commission for the Blind or an independent contractor to perform accessibility testing of a website or an application before launch.

For the authors, one lesson learned from accessibility testing of different websites / applications is: CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) as a security measure that uses visual images based challenge-response is *not* accessible for visually impaired users. This means alternatives to CAPTCHA need to be considered during procurement, solution architecture, software design / development, and ongoing support & maintenance – potentially impacting all aspects of the system development life cycle (SDLC) of a website or an application.

2.2 Multilingual Software Development

Language accessibility, like other aspects of accessibility discussed above, is often an afterthought in the SDLC of a software system. The authors believe this is a poorly understood challenge in most organizations, and multi-lingual software must be planned from the start of a project and then throughout a software system's life cycle. This is so even though modern operating systems for computers and mobile devices support internationalization and have native support for many languages beside English through Unicode and other standards.

In fact, there is substantial experience in the software industry when comes to multilingual software development. Historically, this has to do with "localization" of software for users in different parts of the world. However, as will be discussed below in Section 4.3, this need for "localization" is increasingly in one's own geographic locale, at least for countries with increasingly diverse demographics. In developing multilingual software, the following considerations and best practices are noteworthy:

- *Project Management*.
    - Software features that support multiple languages should be planned early, both in terms of business requirements on target languages and associated budget. This is especially

[2] Web Content Accessibility Guidelines (WCAG) 2.0 provide guidelines on text equivalent content for screen readers: https://www.w3.org/TR/2008/REC-WCAG20-20081211/#text-equiv-all

true for procured systems, because unstated requirements will not be part of executed contracts and, thus, will not be delivered by contractors.
    - To assure timely delivery of software in different target languages, development efforts and translation workflow must be integrated. This is important in most projects / organizations for which development effort and translation effort are managed separately.
- *Solution Architecture & Tools*.
    - Multilingual software means navigation (including the selection of languages) and user interface are in multiple languages -- not just the contents. Elements that support navigation and user interface must be translated, stored, and presented to users under programmatic controls.
    - Multilingual contents mean text, graphics, audio, and video materials in multiple languages. Text and graphic elements require translation; while audio and video elements require, at a minimum, translation of captions or subtitles. These elements must be stored and presented to users under programmatic controls.
    - Developments tools for multilingual development need to be used to reduce repetitive tasks, reduce human errors, and to assure overall quality. An example of this in Linux is the **gettext** package for multi-lingual messages support.
- Team Sensibility
    - Team members should understand that multilingual software development is not “busy work,” because users that require the multilingual support would be unable to use the software otherwise.
    - Optimized user experience (UX) requires language-specific tailoring of elements that support navigation, user interface, and contents. Extra time may be needed for necessary refactoring of these elements to achieve optimal UX.

3 Why is Accessibility Important?

Broadly speaking, we can summarize the importance of accessibility as three points:

- Uncertain regulatory climate - is accessibility the law?
- Better Software Broader user base - does accessibility produce better software?
- Change in societal norm and corporate responsibility - does accessibility produce more inclusive software?

3.1 Uncertain Regulatory Climate

The uncertain regulatory climate, particularly in the United States, has received much attention recently with the lawsuit against Domino's Pizza and subsequent lawsuits against other corporations which have met diverse outcomes. As such, it is not possible to claim with certainty that accessibility is the law, but rather that accessibility may be the law, and the penalties for not following through may be steep.

Accessibility in software products may be required due to laws and regulations and may cause significant risks to IT managers. Certain sectors have specific regulation that governs accessibility requirements, such as Section 508 that covers Federal Agencies or State laws that may require adherence to a standard such as WCAG 2.0 AA. However, the regulatory landscape is rapidly evolving, and recent US case law has applied the scope of the Americans with Disabilities Act to private companies (Domino's pizza v. Guillermo Robles no. 18-1539). As a result, software accessibility may be required for any public software product.

3.2 Better Software and Information Technology

Software that is accessible by virtue of following existing standards is likely to be more consistent and function as intended across more platforms. For example, using our example of alternative text tags (alt tags), images that are correctly tagged produce additional value beyond accessibility, such as improving Search Engine Optimization whereby search engines are able to better categorize an image, providing a fallback for all users before an image loads, and providing tooltip information. Likewise, heading structures which follow a hierarchical structure provide better structure and ease of reading for all users.

3.3 Diversity, Equity and Inclusion

Any level of improved accessibility has the benefit of improving usability and increasing the user base. For example, software which can be accessed by screen readers has the added benefit that visually impaired users will be able to use the product. In this area, there is a wide gamut of accessibility targets to meet, which are usually governed by the standards mentioned previously.

While we have so far focused on disabilities, a broader understanding of accessibility should include the goal of increasing diversity and inclusion. In international marketing, this broader understanding of accessibility is not a novel concept, and localization is commonly used.

4 Managing Accessibility

We can think of Accessibility as three distinct types: exclusionary accessibility, inclusionary accessibility based on standards, and inclusionary accessibility based on diversity, equity and inclusion.

		Type of Mitigation		
		Exclusion	Inclusion: Standards	Inclusion: DEI
Impact	Physical Disabilities	Yes, Separate	Yes, Equal	Yes, Equal
	Non-Physical Disabilities	No	Yes, Depends	Yes, Equal
	Diversity, Equity, Inclusion	No	No	Yes

4.1 Separate but Equal

A common way of improving accessibility is to create multiple pathways to reach diverse customers. This can take multiple forms, such as:

- Text only pages
- Phone support or email requests for assistance
- Degraded service

Text only pages provide an alternative to multimedia rich experiences by copying relevant text to provide an alternative without images, colors, fonts or other elements that may cause accessibility issues to physically disabled individuals. Oregon.gov, the web portal for the State of Oregon implemented accessibility by providing such text only links but phased out this approach in order to improve accessibility[3].

The World Wide Web Consortium's Web Content Accessibility Guidelines 1.0 state that text-only pages should only be used as a last resort:

> 11.4 If, after best efforts, you cannot create an accessible page, provide a link to an alternative page that uses W3C® technologies, is accessible, has equivalent information (or functionality), and is updated as often as the inaccessible (original) page.

The Section 508 Standards make a similar statement:

> (k) A text-only page, with equivalent information or functionality, shall be provided to make a web site comply with the provisions of this part, when compliance cannot be accomplished in any other way. The content of the text-only page shall be updated whenever the primary page changes.[4]

While this may provide a similar end result, it may not provide a similar experience.

4.2 Standards-Driven Approach

The current understanding of accessibility centers on providing physically disabled people with a use experience as similar to that of non-disabled people. For example, in order to be accessible to vision-impaired visitors, a web page typically has certain attributes (such as alt tags, aria labels, headings) which enable screen readers and other assistive devices to interpret and read a page aloud. This understanding of providing a common platform has evolved from the previous understanding of accessibility that allowed a “separate but equal” way of accomplishing accessibility, such as creating text only pages. The unified approach is already an improvement, as when implemented correctly, it does not artificially degrade the user experience for disabled individuals.[5]

As mentioned previously, one advantage of a standards-based approach is that it is easy to test a product against a standard. Numerous tools exist, such as WebAIM or SiteImprove that produce automatic reports in order to highlight areas to be improved. Further, there are specific accreditations such as the Department of Homeland Security’s Trusted Tester Program[6] that allows individuals to become certified in order to conduct accessibility scans to determine compliance with section 508. Through these tools and programs, products can be improved in order to meet existing standards.

4.2.1 Diversity, Equity and Inclusion Shortcomings of a Standards-Based Approach

While a standards-based approach can enable compliance with a standard, there are a number of shortcomings. First, a standard cannot account for specific nuances in meaning, the comprehension level of users, their language and culture or their access to technology and tools. It is true that broad standards requirements can be included, such as ensuring that a website degrades gracefully under low bandwidth situations, but the results can vary greatly in how such a standard is implemented.

A second issue is related to economic equity. Up until now, we have mentioned in passing the use of screen readers to enable accessibility for the visually impaired. While screen readers are a viable tool from a technological perspective, their cost can often times be prohibitive. While Open Source screen

[3] For Oregon’s historical accessibility page, see: https://web.archive.org/web/20060105054832/http://www.oregon.gov/accessibility.shtml
[4] https://www.washington.edu/accesscomputing/are-text-only-web-pages-accessible-alternative
[5] Standards approach to testing, ease of verification, e.g. trusted tester.
[6] For additional information regarding the Trusted Tester Program, see: https://www.dhs.gov/trusted-tester

readers such as NV Access[7] have recently begun to fill the gap, it is important to be mindful that just because a technology is available does not make it broadly accessible[8].

4.3 Accessibility and the Digital Divide: Diversity, Equity and Inclusion Considerations

Using a diversity, equity and inclusion lens can help us adopt a broader understanding of accessibility, which in turn can help us reach a broader audience in an equitable way. When thinking of accessibility through these lenses, it is important to start with what the authors have previously discussed as a "Community of Meaning". A "Community of Meaning" is a construct that embodies the linguistic, socio-economic and cultural space where an individual exists.

4.3.1 Language & Culture

Multilingual Sites. With international travel, migration, and globalization, websites -- together with underlying software and information technology -- often need to offer services to users in different languages from different countries or cultures. For examples, agencies of the state of Oregon where the authors work often provide information in multiple languages. Besides English, these languages may include Spanish, Russian, Chinese, Vietnamese, and others.

Plain Language. While "Language" itself is an important consideration, it is also noteworthy to highlight that education level and literacy also play a role. A common issue seen in Government and large organizations is to have subject matter experts write content, resulting in content that is difficult for the layperson to understand[9]. Mitigations for these types of issues can be put in place in multiple levels, such as conducting user acceptance tests with target audience users that are unfamiliar with jargon or using automated tools to analyze reading levels[10].

Communication Challenges in Multilingual & Multi-Cultural Communications. In this context, the challenges faced are similar to those in international business; except the diversity of languages and cultures may not be distant and is increasingly in one's city, county, state / province, or country. Among professionals that work in international marketing, the typical challenges are [Keegan 2008]:

- A message may not get through to the target audience;
- A message may reach the target audience but may not be understood or may be misunderstood;
- The effectiveness of a message is impaired by noise due to competing messages from other sources that result in confusion, distraction, and miscommunication;
- A message may reach the target audience and understood but may not compel the recipient to take the action desired by the message sender.

Proponents of "one world, one voice" view (with variations such as "one team, one voice") may argue that the world is converging and there is substantial similarity among what appeals to different groups – even if they speak different languages, come from different cultures, or have different socioeconomic backgrounds. At least in international marketing, this view is to be contrasted with proponents of "localization" that believe messages are fundamentally ineffective if they are not adapted or tailored in a culturally meaningful or sensitive way. An important ramification of this view is that literal translation, without appropriate adaptation or tailoring based on cultural and socioeconomic considerations, would be

[7] https://www.nvaccess.org/
[8] The State of Colorado recently amended its Accessibility Law to remove the mention of specific standards and instead focus on the concept of accessibility.

[9] State of Oregon: Department of Administrative Services - Plain language
[10] Hemingway Editor (hemingwayapp.com)

“lost in translation.” This then results in ineffective communications of all types, ranging from loss of fidelity of meaning to total loss of meaning and even unintended misinformation.

To make this point more concrete, let us look at a specific example. During the state of Oregon’s early response to the COVID-19 emergency in the spring of 2020, the state’s COVID-19 website offered downloadable lawn signs with public service messages to encourage people to stay home and minimize contact with others (in order to minimize the chance of COVID-19 transmission). As part of a major communications campaign, the state translated these lawn signs into different languages. However, for recent residents of Oregon that come from countries such as China, the idea of front lawn and the purpose of lawn signs required some explanation, because the idea of a “lawn sign” would be quite unfamiliar. In fact, some staff received the feedback that people that live in apartments or houses without front lawn do not find downloadable lawn signs to be something that they can use.

4.3.2 Economic & Geographical Considerations

Additional considerations are related to Economic & Geographical considerations, such as: Under-developed nations vs. first world countries, poverty and unaffordability, lack of funding and investment, rural areas vs. urban. The digital divide can take a number of forms, such as compute power, types of software used, and bandwidth.

Bandwidth: For example, limited bandwidth may make navigating through complex websites challenging due to load times, incomplete page loads, etc.

Compute Resources: Compute resources is an interesting concept because a device with low computing power and resources will not be able to run certain software products as well as a higher-powered device, or not at all. Issues that may occur related to compute resources are slow loading pages or applications crashing, and a general difficulty using applications or sites. A recent example would be Windows 11, which set certain requirements so high that many otherwise modern devices are unable to install it.

Required Software: For example, open-source software may be free, but may not produce the same results. Certain software may require licenses due to proprietary technology, for example to edit a Microsoft Video or PDF.

Browser Support: Users using operating systems which are End-Of-Life may also be unable to update their browsers, for example Windows XP can only be updated to version 49, and Internet Explorer 8. As such, computers with outdated operating systems or browsers may be unable to render websites.

4.3.3 Opportunities and Challenges

Managing accessibility in a diverse, equitable and inclusive way requires tradeoffs. From our discussion, it should be apparent that stating that software should be accessible is a monumental task. Although the authors believe that an expansive view of software quality is a central component of software quality, with limited resources decisions must be made to prioritize accessibility improvements.

As we mentioned earlier, certain prioritizations may be dictated through laws and regulation, such as the Americans with Disabilities Act or similar legislation. While laws and regulations provide a starting point, expanding accessibility to cultural, linguistic and socioeconomic spaces requires additional considerations in order to make investments where it counts.

The authors believe that these decisions themselves are context dependent and can be decided through discourse in communities of meaning (McDowell 2020). In other words, a group of people has to go through some sort of consensus development which matches the expectations of the community with the best use of limited scarce resources are in order to maximize societal value. These types of decisions will require balancing the limiting factors of effort and cost (of implementation and testing) with what is reasonable accommodation.

In practice, this means that there isn't a one size fits all to an expansive view of accessibility and highlights the issues of promulgating international standards as a one size fits all solution. For example, a community with diverse languages, may choose to prioritize multi-lingual accessibility, whereas a community that serves areas with low bandwidth penetration may seek to optimize content so that it requires low data. In both of these situations the risks associated are similar, that is, services are inaccessible to certain segments of the population, and yet different mitigation strategies may be appropriate.

Conclusion

While accessibility standards provide a good starting point for improving accessibility, they are limited in scope to disabilities and a universal understanding of accessibility. This paper has highlighted some of the shortcomings of this approach and provided alternatives to consider when implementing an expansive and inclusive view of accessibility. While there are many areas that merit additional consideration, such as linguistic, cultural and economic factors, there is no one size fits all accessibility. By working within specific communities of meaning, groups can tailor their understanding of accessibility in order to reach a consensus of where best to implement an expanded understanding of accessibility.

References

"Are Text-Only Web Pages an Accessible Alternative?" Are text-only web pages an accessible alternative? | AccessComputing. Accessed September 13, 2021. https://www.washington.edu/accesscomputing/are-text-only-web-pages-accessible-alternative.

"Department of Administrative Services." State of Oregon: Department of Administrative Services - Plain language. Accessed September 13, 2021. https://www.oregon.gov/DAS/Pages/writingplainlanguage.aspx.

"An Introduction to Accessibility." Digital.gov, December 4, 2019. https://digital.gov/resources/introduction-accessibility/.

McDowell, Jack, and Ying Ki Kwong. "Towards A Culturally Inclusive Software Quality." *Pacific Northwest Quality Conference*, 2020. https://www.pnsqc.org/towards-a-culturally-inclusive-software-quality/.

"Section 508 Trusted Tester Conformance Test Process Version 5." Department of Homeland Security, April 13, 2021. https://www.dhs.gov/trusted-tester.

Warren J. Keegan and Mark C. Green, Global Marketing, 5th edition, Upper Saddle River, NJ, Pearson Prentice Hall, 2008, Chapter 13: Global Marketing Communications Decisions.

Web accessibility. Accessed September 13, 2021. https://web.archive.org/web/20060105054832/http://www.oregon.gov/accessibility.shtml.

Web content accessibility Guidelines (WCAG) 2.0. Accessed September 13, 2021. https://www.w3.org/TR/2008/REC-WCAG20-20081211/#text-equiv-all.

An insight into the life cycle of testing critical security updates supporting large scale security infrastructure

Vittalkumar Mirajkar
Vittalkumar_Mirajkar@mcafee.com

Narayan Naik
Narayan_Naik@mcafee.com

Rakesh Mallinapalli
Rakesh_Mallinapalli@mcafee.com

Abstract

Software security infrastructure requires daily critical updates to help protect against emerging security challenges. Albeit, releasing a timely critical HotFix[HF] update is extremely critical, thorough testing prior to the release, forms an integral part of the release cycle to avoid false positives. This high confidence testing demands a test framework that is machine-driven and can continuously test and release security updates within a limited time window.

The test framework to certify the update should be capable of executing tests on 100's of machines, with varying versions of critical software, supporting all possible OS versions. The entire exercise of initiating machine-driven testing to automated result compilation and analysis needs to complete within a stipulated time frame for release readiness.

Keeping such a mission-critical, high availability environment operational and triggering test for validation followed by a quick turnaround of the large result analysis to certify release readiness, is a continuous complex firefighting exercise.

Through this paper, we present a case study that demonstrates a framework, tools and techniques that support the demanding nature of rapid security patch testing. This test framework enables quick, consistent, and reliable methods to meet an ever-changing threat landscape.

Biography

Vittalkumar Mirajkar is a Software Architect at McAfee, with 15+ years of testing experience ranging from device driver testing, application testing and server testing. He specializes in testing security products. His area of interest is performance testing, soak testing, data analysis and exploratory testing.

Narayan Naik is a Software Engineer at McAfee, with 11+ years of experience in exploratory testing and performance testing. He holds an expertise in providing consultation to enterprise customers for features and compatibility of various security products and security solutions deployed. His areas of interest are inter-compatibility test areas, performance testing and encryption product lines.

Rakesh Mallinapalli is a Software Engineer at McAfee with 5+ years of experience in building full stack automation frameworks. He is a passionate developer who has used ML and Data analysis to build highly efficient CI/CD test beds.

1. Introduction

Maintaining computer security of large-scale infrastructure demands a daily assessment and patching for latest vulnerabilities through HF deployment. Timely availability of the critical patches plays a major role in this. HFs are released in extremely tight schedules. This requires a much shorter testing time. Supporting timely testing and release of HF patches across a wide spectrum of critical software products is a complex moving puzzle. You need a best-in-class Continuous Integration/Continuous Delivery (CI/CD) framework to do so which can support daily test and release cycles.

2. What we want to test

Our goal is to test and certify each critical HF patch for daily release readiness. The release readiness is a to be completed from build availability to release certification within a tight time schedule of a couple of hours.

These could be a single patch or multiple patches that need to be tested on multiple product versions installed on a spread of supported OS's. To understand the complexity in the number of products, operating systems, and environments we are required to certify against, consider the following

Product set *Pn = {P1, P2 …. P12}*

Each Product further has "m" versions which are active in the field, there by the product version(Vm) set is Vm *= {V1, V2 …. V10)*

We could restrict the per product version active in production to 10

PnVm = {P1V1, P1V2… P2Vm, P2V1, P2V2… P2Vm, …….P10V1, P10V2… P10Vm}

Each element in *PnVm* is to be tested on supported OS's.

OSn = {OS1, OS2, OS3…. OSn}

If we consider the various stages an OS is present from its release till EOL, each OS itself has multiple flavors and multiple patch versions active in the field.

Example:

Windows 10 has had 12 releases (Windows_10_Versions), each version having 4 flavors supported (Microsoft, Windows 10 editions 2021) i.e Windows 10 Home, Pro, Pro for Workstations, Business.

Considering just Windows released versions till date, Windows have had 15 major releases (Microsoft, Operating System Version 2020)

In our case this was

Pn = 1 to 12

PnVm = 1 to 10

OSn = 1 to 15

Taking 10 major versions per OSn release, have 4 flavors each

OSnVm = OS Version released

OSnVmFx = Flavors active per version

OSnVm, where m range is 1 to 10 and *OsnVmFx*, where x range is 1 to 4

Total number of combinations, theoretically possible is

*Pn*Vm*OSn*Vm*Fx = 12*10*15*10*4 = 72000*

Note:

Not all OS's releases have same number of released field versions (examples: OS1 may have had 2 releases, whereas OS2 may have had 5 releases.)

Not all Products have same version of releases (example: P1 had 1 to 10 patch releases, where P2 had 1 to 8)

2.1 Data sanitization:

As the 1st step creating a workable set, we need to identify which data set is relevant. Visualizing the Product vs OS supported as a 2-dimension table helps us.

	OS Supported Set														
Product Versions	OS1	OS2	OS3	OS4	OS5	OS6	OS7	OS8	OS9	OS10	OS11	OS12	OS13	OS14	OS15
P1V1															
P1V2															
P1V3															
P1V4															
P1V5															
P2V1															
P2V2															
P2V3															
P2V4															
P2V5															
P3V1															
P3V2															
P3V3															
P3V4															
P3V5															
P4V1															
P4V2															
P4V3															
P4V4															
P4V5															
P5V1															
P5V2															
P5V3															
P5V4															
P5V5															

Plot the Product Version vs Supported OS as a Table

PnVm vs OSnVmFx

This data at the start is all filled with Unknown values.

A simple db is to be built to store this data set

Rule for sanitizing the relevant data:

1. Not all product versions are supported by all OS's.
2. Latest version of product series / versions is not supported by earlier version of OS's. This information helps in optimizing the product vs OS supported versions.
3. Installation count of specific older version on older OS's could be insignificant and help us ignore it if the total installation count is insignificant.

2.2 Relevance of data:

Following channels helps us pick relevant data

1. Product Installation telemetry
2. Product install counts from Product Team / Product Managers
3. OS Market share (Statcounter 2021)

	OS Supported Set														
Product Versions	OS1	OS2	OS3	OS4	OS5	OS6	OS7	OS8	OS9	OS10	OS11	OS12	OS13	OS14	OS15
P1V1															
P1V2															
P1V3															
P1V4															
P1V5															
P2V1															
P2V2															
P2V3															
P2V4															
P2V5															
P3V1															
P3V2															
P3V3															
P3V4															
P3V5															
P4V1															
P4V2															
P4V3															
P4V4															
P4V5															
P5V1															
P5V2															
P5V3															
P5V4															
P5V5															

Once the relevant data is available, we apply 80-20 (Armand Ruiz Gabernet 2017) of data analysis to define a product vs OS relevance for test prioritization.

The prioritization is defined as

OS deployment relevance * Product installation relevance.

One this rule is applied, the new optimized test set appears similar to data presented in the table.

3 Building the test bed:

We need Continuous Integration / Continuous Delivery test setup which meets the following specific requirements

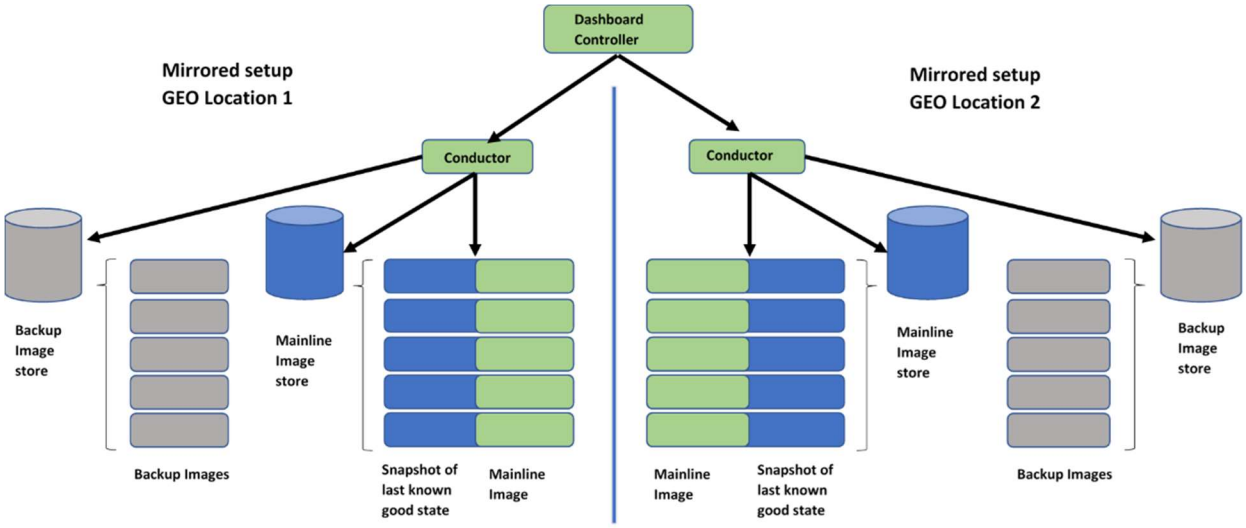
1. High availability
2. Has 2nd level redundancy setup
3. A status monitor dashboard
4. Automated Build release to test readiness infrastructure

3.1 Building beyond 2(N+1) Level Redundancy

We borrowed concepts of data center power supply 2(N+1) (Data Center Redundancy: 2N, N+1, 2(N+1) Explained 2019) redundancy and build a test infrastructure to go beyond 2(N+1) redundancy to maintain the availability of test machines.

Mirrored test infrastructure is setup in two different geos to achieve 2N redundancy. The mirror sync is triggered every 12 hours to maintain same operational state.

In each of the setup, each operating system image state can be reverted to a known good state if the test needs to be retriggered. This helps us achieve within a given setup N+N redundancy. In case the test image is corrupted, the monitoring dashboard has a 2nd level redundancy failsafe which kicks in to trigger machines from the 2nd backup rig for testing. This makes it a 2(N+N+N) redundancy setup.



3.2 The Test execution controller:

We need a one stop controller which helps the user understand the current state of test execution as well as to auto trigger any remediation required, making an engineer's manual intervention the last resort. The controller dashboard needs to meet the following requirements.

- Front end which visualizes an easy-to-understand current flow of execution.
- Calls out current run state and failure (if any) and steps taken to remediate it.

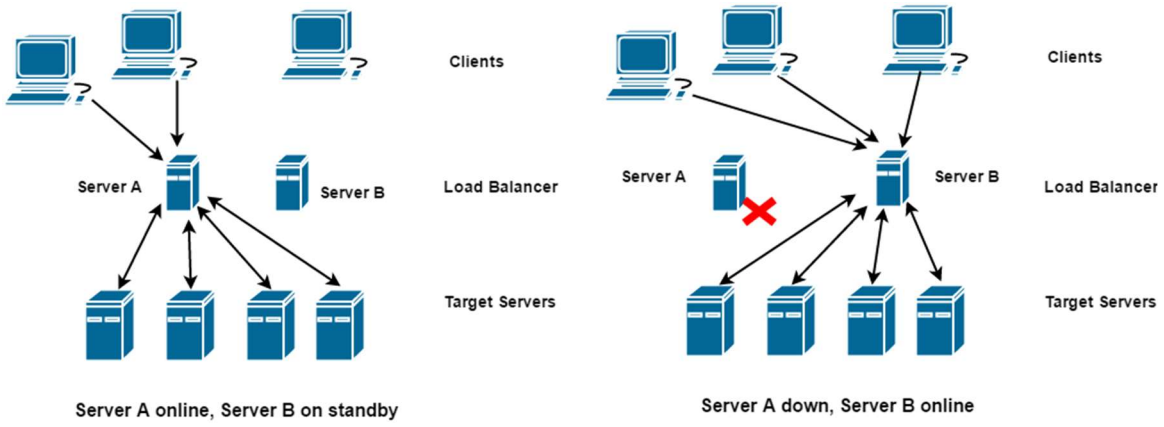
The test controller needs to be backed by an intelligent backend which understands current state and kicks in remediation as and when required. One of the important steps in this chain is Power on Self-Test (POST)

POST is done to check test setup readiness to trigger validation cycles.

This includes (but is not limited to):

Failure Points	Remediation
Network connection	Network card reset followed by reboot
CPU spikes	Poll back followed by reboot
Hard disk availability	Release Hard disk using VMware api's

Test Controller is the next important part of this setup. Test controller is built on an Active-Passive model (Raghumb.gitbooks.io 2021) to ensure high availability.



Active Passive Configuration
(Eg Load balancer)

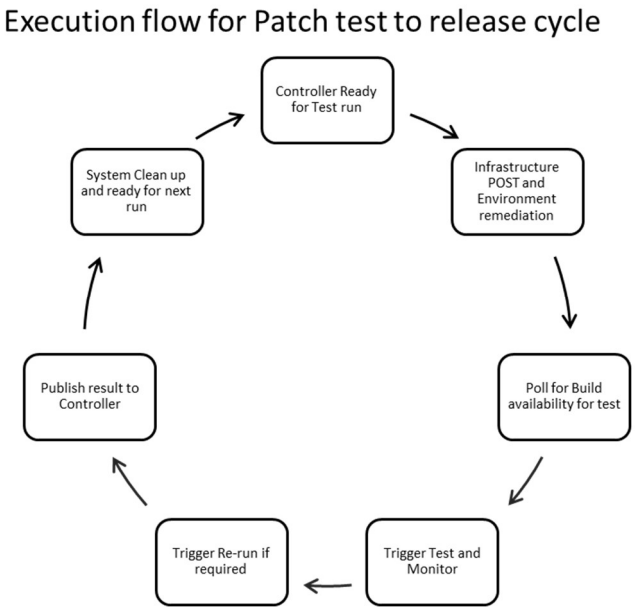
3.3 The Hardware

We have used Dell Blade server (Dell PowerEdge 2021) with 16 CPU rig for processing power and for storage we have used Dell-EMC XtremeIO (XTREMIO_X2 2020) two brick solution.

VMware vCenter and vShpere is used as the backbone infrastructure.

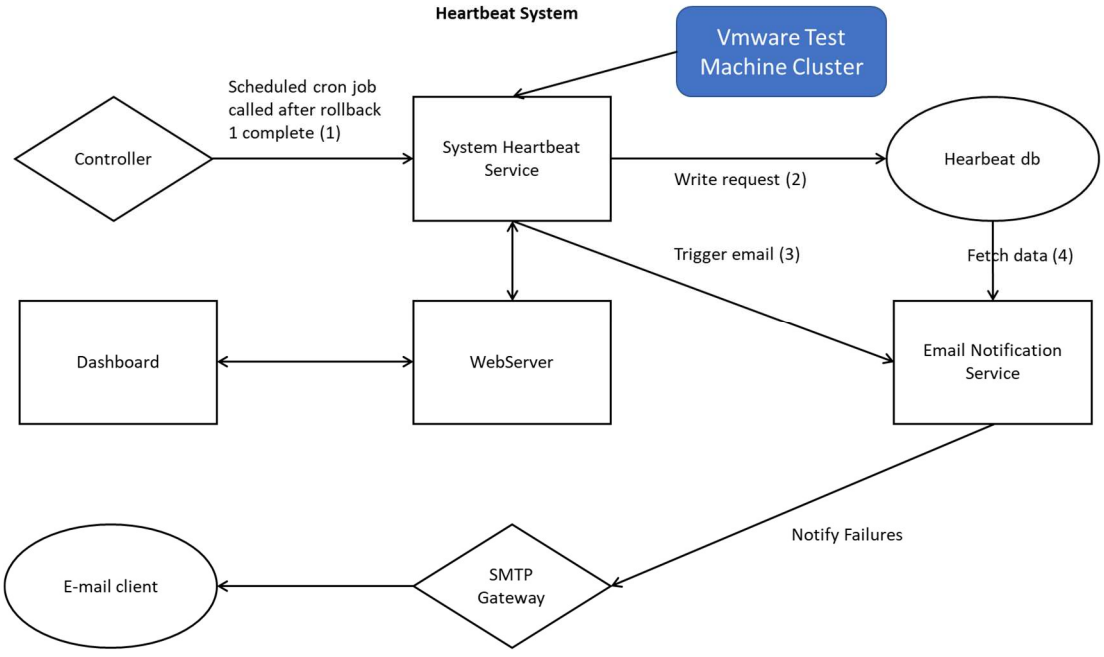
4 The execution flow

Patch build availability to release testing follows the below flow for automated test to release readiness cycle.



At a deep dive level following are the step wise process.

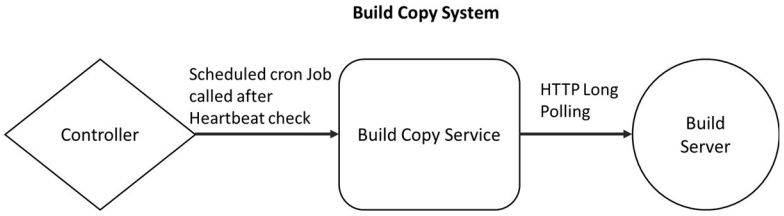
Step 1: Heartbeat System



Prior to checking for build availability for testing, working state of the test environment is confirmed. Test controller triggers “Heartbeat Service” which checks for test environment machine’s health check. This checks VM node cluster is in working condition (ex: Availability, reachability, and disk availability). This information is stored in “HeartBeat db” for future reference and an email notification is sent to stake holder about the base state of the test environment. Visually, this data is also present in Test Controller dashboard as well. The system has the intelligence built in to check environment integrity, take remediation action

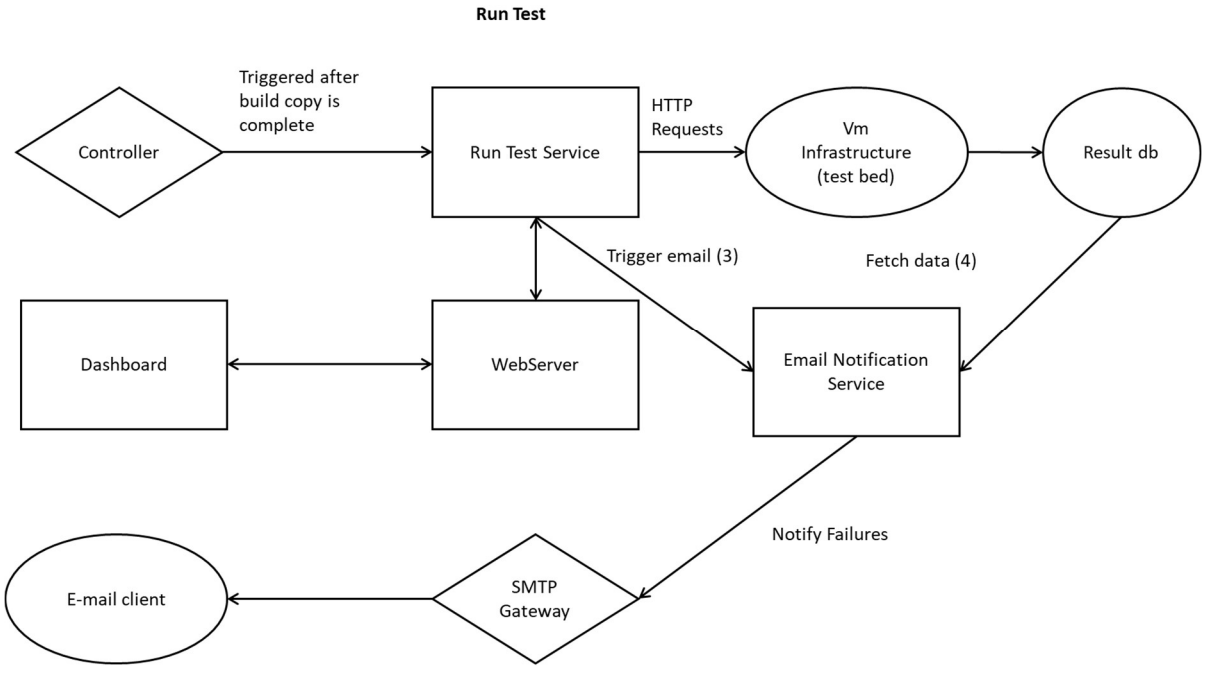
Once the test environment stability is confirmed, build copy system is triggered.

Step 2: Build copy system



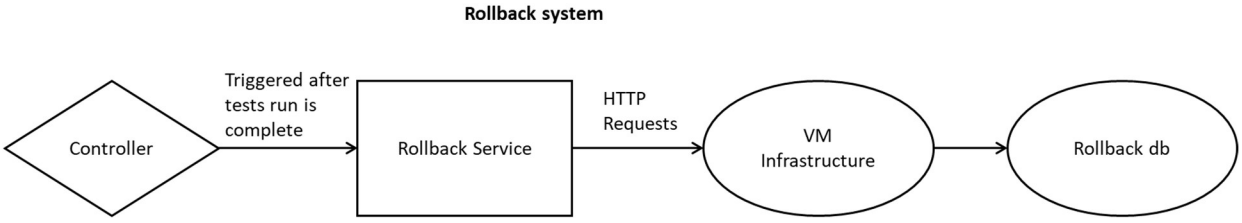
Test controller periodically does a https long polling to check for build availability in GIT repository. As soon as the build is available, it is copied to local controller repository and next step of test execution is triggered.

Step 3: Run Test



Controller triggers tests on the machine set hosted in VM infrastructure for sequence of tests that needs to be validated to certify the build. The results are stored in “Result db” and simultaneously test status is updated in dashboard front end. An email notification is triggered to the test owners. The test run controller has intelligence to take remediation action and intuitively trigger re-runs in case of test failures.

Step 4: Rollback system



Once the test runs are completed and results are availability either to block the release or pass the build, the results are archived, and the environment is scrubbed to be ready for next test cycle within 30 minutes of last test completion.

Rollback service handles infrastructure services using VM api's and does following jobs,

- Image roll back to known good state
- Snapshot (in case an issue detected in previous test run) and roll back
- Replace the images (if corrupted by previous test runs)
- Retained bad nodes for future investigation and fresh copies made from gold images for next run

For one end to end cycle, it is a 6hrs round trip time including POST and environment scrubbing posttest completion.

5 Test Results

Each stage has individual results published as email report, stored in reference db and also updated in live status dashboard.

Heartbeat service:

Heartbeat service publishes results of the test bed health check and remediation steps taken if any. This gives a summary final countdown system check for next run.

Mid run status from dashboard:

Below is the screen shot from dashboard when test is still in progress. A real time view gives test run status, not just over all test status but each individual module test status as well. This helps us isolate failure modules when a system reports test run as failed.

MOVE VMs	UPD	ODS	OFFICE	STATUS	RBS	Disk_Space(MB)	Warnings/Alerts
20H2X64-MOVE49					-	-	
20H2X32-MOVE491					-	-	
10RS264-MOVE461					-	3115	
W81P64-MOVE46					-	418	
10R4X64-MOVE47					-	12620	
2K16-MOVE47					-	3724	
W81P64-MOVE47					-	11134	
2012R2ST-MOVE45					-	15415	
10RS5X64-MOVE48					-	44494	
2K19S64-MOVE481					-	-	
10RS586-MOVE481					-	-	

Test completion:

Below is the screen shot when the test run is complete for one of the products, among multiple products which is a being tested with this system.

MAC Machines	UPD	ODS	OAS	STATUS	RBS
MAC_OS_X_MOJAVE_CORP2					-
MAC_OS_X_BIGSUR_CORP-MEDIUMDAT					-
MAC_OS_X_HIGHSIERRA_CORP					-
MAC_OS_X_CATALINA_CORP-MEDIUMDAT					-
MAC_OS_X_CATALINA_CONS_NE					-
MAC_OS_X_MOJAVE_MEDIUMDAT-MEDIUMDAT					-
MAC_OS_X_MOJAVE_CONS					-
MAC_OS_X_BIGSUR_CONS					-

The test framework in current state is catering to 6 products. Each products test bed consists anywhere between 8 OS's (as shown above) to 200 OS's. The complete test run turnaround time is of 6 hrs.

6 Environment upkeeping challenges:

The major two buckets for environment upkeeping are revisiting relevant product versions for testing, and patching of OS

6.1 Relevant Product versions for testing:

Products released to production themselves go through an increase/decrease in deployment numbers. This needs to be reflected in the testbed used for testing. A quarterly review on the deployment numbers sourced from telemetry, product deployment numbers and Product Program Managers help us scale up or scale down a product test set.

6.2 Patching OS's:

Test bed needs regular upkeeping to maintain sync with Microsoft's Patch Tuesday's (Patch Tuesday 2005). This is required to reflect the changing end customer environment due to OS patching/upgrades. OS's patching is also carried out in an automated fashion with one rig going in for an upgrade while the 2nd back up rig being used for main line testing. This ensures zero downtown during system upgrade cycles.

7 Take always

Rapid test-to-release cycles for automated testing of critical security updates for various combinations of products and OS can be achieved. Some of the areas where this model of testing can be used are,

- For quicker certification of HFs for release readiness.
- For releasing minor fixes quickly to field rather than waiting for monthly release cycles.
- Incremental improvement fixes can be tested and released to production as and when feature is ready and not wait for major release cycles.

This approach helps us go one step closer to achieving a true CI/CD flow for this testing.

References

Armand Ruiz Gabernet, Lead Offering Manager, IBM DSX & WML, Jay Limburn. 2017. *Breaking the 80/20 rule: How data catalogs transform data scientists' productivity.* August 17. Accessed June 28, 2021. https://www.ibm.com/cloud/blog/ibm-data-catalog-data-scientists-productivity.

2019. *Data Center Redundancy: 2N, N+1, 2(N+1) Explained.* May 9. Accessed July 10, 2021. https://www.123.net/data-center-redundancy-2n1/.

Dell PowerEdge. 2021. *PowerEdge M1000e Blade.* Accessed May 10, 2021. https://www.dell.com/ba/business/p/poweredge-m1000e/pd.

Microsoft. 2020. *Operating System Version.* Sept 15. Accessed June 20, 2021. https://docs.microsoft.com/en-us/windows/win32/sysinfo/operating-system-version.

—. 2021. *Windows 10 editions.* Accessed June 10, 2021. https://www.microsoft.com/en-us/windowsforbusiness/compare.

2005. *Patch Tuesday.* July 6. Accessed July 23, 2021. https://en.wikipedia.org/wiki/Patch_Tuesday.

Raghumb.gitbooks.io. 2021. *Active-Active Active-Passive configurations.* Accessed July 10, 2021. https://raghumb.gitbooks.io/a-guide-to-software-architecture/content/infrastructure_concepts/active_active_active_passive.html.

Statcounter. 2021. *Operating System Market Share Worldwide.* July 1. Accessed July 15, 2021. https://gs.statcounter.com/os-market-share.

Windows_10_Versions. 2015. *1.* Nov 14. Accessed June 10, 2021. https://en.wikipedia.org/wiki/Windows_10_version_history.

XTREMIO_X2, Dell. 2020. "XTREMIO X2." *https://www.delltechnologies.com/asset/en-za/products/storage/technical-support/h16094-xtremio-x2-specification-sheet-ss.pdf.* Accessed July 10, 2021. https://www.delltechnologies.com/asset/en-za/products/storage/technical-support/h16094-xtremio-x2-specification-sheet-ss.pdf.

Testers Have Requirements Too!

Moira Tuffs

moira.tuffs@3dsystems.com

Abstract

Requirements from the testing team are often overlooked when business or user requirements are the focus of a program. Business or user requirements contain quantifiable objectives such as speed improvements of 30%, or the user should see plain English error messages. However, there are underlying assumptions that make these business specifications measurable. Using the speed improvement as an example - where is the 30% measured from? What is the baseline to show this requirement has been met?
Therefore there is a new demand for documented requirements from the testing team. In order to show a 30% speed improvement, the testing team must first establish a baseline and be able to measure the improvement against that baseline.

In this paper I will highlight the testing requirements based on familiar business or user-based requirements, along with some of the lesser requirements when the non-standard paths through the application are considered.

By the end of this paper, you will have the essential tools to critically examine what types of testing requirements should be added to enhance the business or user requirements relevant to your work, thus resulting in a more complete set of requirements. And we all know that complete requirements are the starting point to a great product.

Biography

Moira Tuffs is the Manager of the Embedded Software Quality Assurance team at 3D Systems. For more than 30 years she's been involved in software and hardware products on both sides of the Software Engineering and Quality Assurance fence, gravitating to the QA side for the last 12 years. She has experience in projects from a wide variety of domains including space craft control systems, signal integrity and PCB design systems, as well as embedded systems.

Her career goal is to push the Quality and Reliability message into every aspect of product generation, such that Quality Assurance becomes everyone's responsibility to attain and police.

1 Introduction

This informative brief has come about from a simple question I posed at the beginning of testing our project on an existing printer. The question was, “How do we show evidence that the home position is calibrated for the print platform when we select the calibrate button on the GUI?” Another way in which to word this question is, “How do we know the button does what it is intended to do?”

For some background on this particular project, this printer started with some hefty marketing requirements for speed and reliability improvements. The customers of 3D Systems need their printed jobs to be created correctly, quickly, without need of repeats, and the printed parts must be dimensionally accurate. Not all of these specifications are within the testing roles of the Software Quality Assurance (QA) team; however, our job as QA is to make sure the firmware and hardware deliver to the requirements requested for the program. We do our best to relate to our customers or, “put our customer hat on” if you will and test the product like a customer would. Not only does this mean that the products we sign off on are user friendly, but it also means that when customers come to us with problems, we have likely already walked through such a scenario in our own testing. This process enables our team to deliver a better product that we are confident will meet the customer’s expectations.

As a team we dug into the information we needed to answer the original question, and the conclusion that we collectively came to as the QA team became, “We have requirements too!”

2 Why do we need testing requirements?

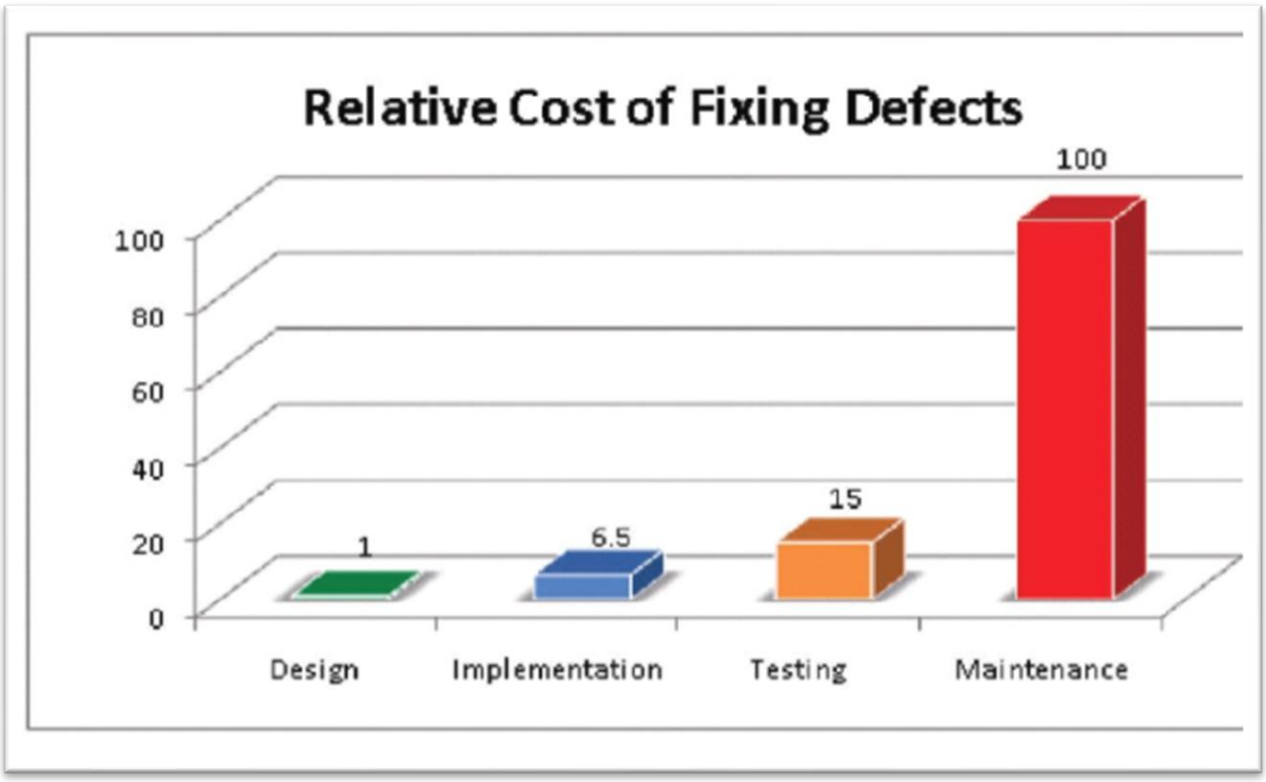
Testing of embedded systems comes with its own unique set of problems stemming primarily from the use of a closed system. Unfortunately, one of the headaches we find ourselves dealing with time and time again in the position of Quality Assurance is the ability to show evidence of testing. Log files can provide a wealth of information that may be useful to the developers. But they often have cryptic debugging information which you then need the keys to interpret them. Additionally, log files have not always been targeted as a QA resource, and they are often an afterthought to be a tool which the QA team can use. This creates a disconnect between what developers are producing and what the QA team can see of that product.

For example, sometimes the developers flood the log files with data, and it becomes a tedious scavenger hunt trying to identify the specifics of a task; when a task started or ended or when a set position was reached by a mechanical component before moving off again.

There are other times when the QA team finds a cryptic message: “Sent ack”. Sent to who? Did they receive it? What message was the “ack” associated with? There are many questions that are based off this one clue, creating unnecessary clutter that now someone in QA has to spend time and resources diving through to answer a part of the bigger question. While precious time is being spent on a scavenger hunt of sorts, there still remain deadlines and testing expectations that need to be met.

When put in such a position, it becomes important to define a set of testing requirements to hold ourselves and others accountable, not to mention avoiding unnecessary confusion in the future. Defining a set of testing requirements not only helps with testing down the line, but it also helps get the testing team in on the action earlier and starts the conversation flow.

Below shows a chart from the Journal of Information Systems Technology and Planning, paper by Dawson et al. showing the relative cost of fixing defects.



The results of the research performed by the IBM team have often been attributed to an increased focus on the upfront requirements and design phases. Adding testing requirements forces the testing thought process up stream thus saving all of us another potential lecture to ensure bugs can be found early on in the product development life cycle. It is generally accepted industry practice that complete clear and concise requirements will lead to an overall more complete product (Kar and Bailey 1996).

3 What types of testing requirements should be considered?

The types of testing requirements will vary for each problem domain and application, be it a simple software program with few paths through, to the complex embedded systems used with 3D printers. Requirements can also be resource driven as well as data driven. The following are highlighted as general areas to be considered; however, the overarching theme is to approach each program requirement with the same question – What information will the QA team need to be able to verify this requirement has been met at the end of the program?

3.1 Requirements that will allow the QA team to collect evidence of testing.

Let's start by going back to the log file example. Often in the world of 3D printers some part of the printer needs to move to a set known position. For example, the build plate moves to its lowest position to begin printing its first design. The question we now need to ask is, how do testers know that a specific position was achieved? From there of course there are more questions that can be answered. In fact, the more detail that testers give on the products they are testing, the more evidence there is to use in future

problem solving. Therefore, we ask ourselves, where is that evidence that you can post to a testcase to prove the results with evidence of testing?

In this specific problem, the answer to that second question is in the log files. The testing requirement would be to log the expected set position, alongside the actual position achieved, in units that make sense for the types of procedures that the user can perform.

How many times as a tester have you set a timer on a watch or a phone to track the time an operation takes? Wouldn't it be helpful to have a summary of that operation once it's completed in the log files?

Another solution to problems we face lies in the functions that our customers would also like to see in our products. There are some functions such as printing that the customer would like to anticipate when the task will finish, so they can plan any resource usage appropriately. The testing task for a print job is to verify reported start and end times are correct. Another testing task is to evaluate the accuracy of the estimated total print time versus the actual total print time. In the log files start and end times are reported, but no overall print time. That drives my QA team crazy, because now they must search the logs to match the start and end job names, note the timestamps, and do the math to answer the question of how long a build took to complete and how accurate the estimate was based on the total build time.

Testing Requirement: At the end of the operation report a summary of time start, end and duration.

3.2 Requirements for testing improvements from a Baseline metric

Releasing updated versions of printer firmware often includes increasing the speed to certain processes. For this example, we can go back to the original project that sparked the idea for this paper. Part of the requirement was to have a target X% speed improvement for printing. The first question to ask here is, what is the baseline? Without a baseline there is nothing to measure speed against, and therefore any increase or decrease that we make to the speed of the processes is irrelevant to the requirement. More examples of questions the QA team could ask next include: What material? What print job are we looking at specifically? Is it every print job that must improve? Can we average the print jobs to get an overall percentage improvement? The devil is in the details for this requirement, and the QA team is often the one to see the need for those details to perform their job function fully.

In this specific instance, the engineering team had already identified some processes that were more than ready for speed improvements. Great – testing this speed improvement should be simple. But come back to that question – what is the baseline? And remember the customer runs the printer as a system, they are not interested in the separate processes that combine to make the overall print time. So, putting on our "customer hats" so to speak, the customer will notice if their print job times are faster or slower, not that a certain piece is moving a fraction of a second faster than before.

Testing Requirement: Specify the set of print jobs, and the specific material for those print jobs to validate the speed improvements.

3.3 Requirements for testing the customer intent system.

Take a moment to consider what requirements there might be for testing the full product version of the hardware and software, and to be able to test the full system. On one occasion the QA team that I work with were asked to validate an updated version of a hardware part for the printer. We were told that no software change was needed to support this hardware change.

The purpose for the updated hardware part was to make a cleaning process more effective using a different wiping component. Unfortunately, the footprint for the updated part was larger than the existing part by only a few millimeters. This tiny change was not considered to impact any action that the user or software would take when interacting with the cleaning process.

Even with the previous hardware part, one step of the cleaning process instructs the user to remove the updated part for cleaning itself. The QA team found the new hardware part extremely difficult to remove as the location of a cross beam was now holding the new part in because of those few millimeters in extra length. The mechanical team had tested the removal and insertion of the part, but not as part of the full customer intended process. Just a few millimeters were what impacted the ability for a customer to clean the product. Thankfully, there was a specific test for the complete cleaning process. Therefore, a software change to move the crossmember before the cleaning process started was implemented, and tested, to ensure the customer intended process would go smoothly.

In early phases where a project contains mechanical, electrical and embedded systems, not all the parts (hardware, electronics, firmware and software) are available as a single testable system. Meaning that there may be further requirements on testing with a simulator or emulator environment that will depend on the individual situation. However, for most of the projects, testing the customer intent system is a critical component of the testing cycle.

Testing Requirement: Ensure hardware and software are updated as a system for regular testing cycles.

4 How to get the testing requirements into the project.

Programs have a defined content, and the schedule and end delivery date are driven from that content. As such, it is very important to get the testing requirements in early so they can be scoped alongside the development work and be incorporated into the schedule. In a continuously changing environment testing requirements would be added as stories and tasks to the backlog, to be pulled into the planning for the sprint cycles.

However, in most cases, the program requirements are defined in the Marketing requirements document. This document identifies the customer's need and possibly the business' case for the requested feature or product. A typical program life cycle has the engineering team then respond to the Marketing requirement, with their set of features and identified use case scenarios. In our agile world, these are listed in the JIRA project to create a complete backlog.

It remains advantageous as the QA lead on a program (or the QA manager) to review early drafts of the Marketing Requirements to build your understanding of the types of requirements that need firmly defined baselines. This provides a streamlined process that then can be organized properly to fit schedules and deadlines as well as bringing the QA team into the conversation of requirement documentation.

As the engineering team on a project is digging into these program and technical requirements, it is advantageous to find some allies on the development team. Working closely with the scrum master or technical program manager to make sure the testing requirements are understood reduces the risk of miscommunication later in the process. Also, scheduling some time in one of the planning meetings gives an opportunity to educate the engineering team as to some specific requirements the QA team have to enable them to show evidence of testing.

As the project then moves into the design phase, I recommend pairing a QA and Software Engineer to walk through the use cases. At this point it is important for the QA engineer to ask the very pointed questions, "How can we test what you are planning to implement?" and "What data will be logged as part of this operation?" and "Can you make this information simple to find?". These are important to ask given

the requirements that were explained in detail earlier. Together they should be able to identify the information needed to show testing evidence and allow the QA team an easier task of locating the specific information needed to report as evidence of testing. The output of these types of collaborations will become additional stories for the development team – however, they will be simple for the QA team to test!

Let's step through a simple example to elaborate on this process.

Marketing Requirement: User should be able to see the completed print time on the UI at the end of a print job

The breakdown of this requirement into architecture and design is outside the scope of this paper. Taking a shortcut, it's not hard to end up with a set of JIRA items describing the development work as below.

JIRA Story: UI needs to display print end time when a print job completes

JIRA Story: Control code should report print job end time to the UI when a print job

The QA team can now ask the question how do they validate the end time on the UI? Which leads to the more in-depth question about the control code, and how that data point can be validated. Having a QA team member work the development team member on the control code story should conclude that additional information is logged at the end of a job in a format and log file that the QA team can find easily. Using keywords or tags is often a neat solution for these scenarios. At some point a JIRA item should be added for the additional work required to help the QA team verify the above JIRA items. An example of such an item could be

JIRA Story: At the end of a print job, the control code should record in a log file:

- Print start time
- Print end time
- Print duration

With the implementation of this additional JIRA item the validation of the original marketing requirement becomes an exercise in log file collection, search and matching the data to that shown on the UI.

5 Conclusion

Having a complete set of requirements for consideration at the start of a program is the central theme to this paper. The QA team is often the last to get a voice in this requirements process, which can result in a struggle to have the right tools and data to show sufficient evidence of testing. Making a well-informed decision on product readiness to ship has been impacted by missing early testing requirements. The goal of this paper has been to suggest some options and techniques to change that.

In my experience, the QA team need to be involved early in the design of the project to review requirements for testability and suggest additional functionality to aid in the QA testing process. The

updated study from Dawson et.al is as relevant today as it was in 2010. Finding bugs early in the design process is a cost savings for any program. Well defined requirements from all roles in the team will lead to a cleaner overall program that benefits all parties.

Showing evidence of testing from engineering log files has long been problematic for the QA team. Identifying those features where the information in the log file would be useful to the QA team in proving a feature has been fully implemented and is working as expected, is a step to adding testing requirements into the development life cycle for the program. Giving the QA team a voice in that early identification adds the formal request for work to be included in the scoping and planning process, and clearly states the need for detailed information for QA to complete their task.

Identifying the marketing or program requirements where baseline information is needed to prove that requirement has been met shows another option to include the QA team early in the process and allow a complete set of requirements for the team to consider when gathering the full scope needed for implementation and test.

References

Dawson, Maurice & Burrell, Darrell & Rahim, Emad & Brewster, Stephen. (2010). Integrating Software Assurance into the Software Development Life Cycle (SDLC). Journal of Information Systems Technology and Planning. 3. 49-53.

Kar, Pradip and Bailey, Michelle, "Characteristics of Good Requirements", *Proceedings of INCOSE 1996, Volume II.*

Acknowledgments

I would like to thank my reviewers Vanessa Tuffs, Amith Pulla, and Philip Daye for their unending patience, words of wisdom and encouragement. Without their help, this paper would still very much be in bullet points without a complete sentence in sight.

The Affordances of Quality

Steve Upton

steve.upton@thoughtworks.com

Abstract

The phrase “build quality in” has become ubiquitous in the software industry. Many books, articles and papers propose specific examples of building in quality, but what is the common thread that ties these approaches together? How should someone looking to “build quality in” approach the problem at a high level?

This paper explores the idea of affordances, from the field of Human Centered Design, and how they can be used to identify places where quality can be built into the process.

Biography

Steve is a Quality Analyst at Thoughtworks who sees his job as helping to build teams capable of reliably delivering quality software. He works to support and empower teams with all they need to take ownership of quality.

Steve has worked on a wide variety of products, from mainframes to microservices and enjoys bringing new perspectives to different technical contexts. He has a particular interest in complex socio-technical systems and how we can work with them.

He is passionate about systems thinking, building quality into culture and testing as part of continuous delivery in modern, distributed architectures. Outside of work, Steve enjoys travel and mountains.

1 Introduction

The phrase “build quality in” has become ubiquitous in the software industry. Many books, articles and papers propose specific examples of building in quality, but what is the common thread that ties these approaches together? How should someone looking to “build quality in” approach the problem at a high level?

When describing how to enact change in an organization, David Marquet rejected “directed programs”, which seek to enact change in a top down manner. Instead, Marquet “searched for the organizational practices and procedures that would need to be changed in order to bring the change to life with the greatest impact” (Marquet, 2001). In essence, Marquet was searching for places to ‘build in’ the changes he wished to see in his organisation.

The concept of affordances, from the Human Centered Design movement can be used to identify where such changes can be made.

2 Affordances

In The Design of Everyday Things, Don Norman introduces the idea of an “affordance”, which he describes as “...a relationship between the properties of an object and the capabilities of the agent that determine just how the object could possibly be used. A chair affords (“is for”) support and, therefore, affords sitting.” (Norman, 2013).

A commonly referenced example of affordances at play is the design of doors. Some doors, such as the one depicted in Figure 1, have flat surfaces that can only be pushed. This door is said to have an affordance for pushing because its design only allows that interaction (pushing).

Figure 1. A door that affords being pushed.

Not all affordances are intentional or lead to positive outcomes. A flat surface affords having objects placed on it, which can lead to flat surfaces accumulating mess.

Affordances “determine what actions are possible” (Norman, 2013) and are a key part of the Human Centered Design movement. Affordances also exist in our processes and ways of working. This paper will explore the ways in which building these affordances can improve our ability to deliver quality software.

3 Remove the “QA” column

Many teams use a Kanban board to visualise their work and progress, such as the board depicted in Figure 2.

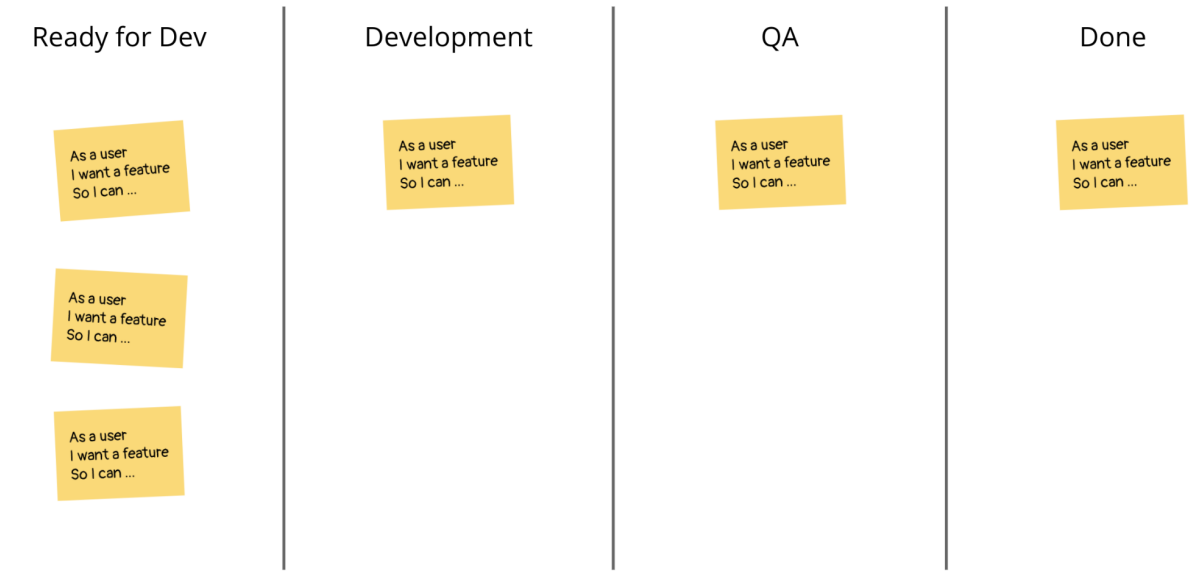


Figure 2. A Kanban board.

Every team’s board is different, influenced by the framework or methodology they are using (eg. Scrum or Kanban) and the needs of their stakeholders. Despite the diversity of board setups, a common feature of many boards is the presence of “Development” and “QA” columns.

The presence of a traditional QA column on a board has a number of issues:

It encourages and increases the number of handoffs. The titles “Development” and “QA” map neatly to the job titles “developer” and “QA”, subtly implying that each column is the domain of the respective role. Poppendieck (2003) identifies handoffs (such as those between developers and QAs) as a source of waste, as tacit knowledge is left undocumented or not passed along.

It allows quality concerns to be deferred. Even if all people involved understand the need to build in quality, the presence of a separate “QA” column implies that “quality” can be taken care of at that stage.

In the same way a flat surface affords placing objects on it, a separate “QA” column affords pushing quality and testing concerns later in the process.

Elisabeth Hendrickson describes a situation in which investing in a large test team to test software separately from the development team resulted in more bugs being released to users. “The developers here thought that the independent test group had assumed full responsibility for all the testing”

(Hendrickson, 2001), which resulted in them investing less time in test efforts themselves. In this case, the large test team created a clear affordance for the deferring testing until later in the process.

Kent Beck describes a similar relationship between quality and a separate quality team, “Having a separate quality department sends the message that quality is exactly as important to engineering as marketing or sales. No one in engineering is responsible for quality. Someone else is. Putting QA as a separate department within the engineering organization also sends the message that engineering and quality are separate, parallel activities.” (Beck et al, 2004).

Removing the column from your board sends a clear message that quality must be built in during development and reframes the Developer-QA relationship from one of handovers to one of close collaboration. Removing the “QA” column creates an affordance for building in quality.

4 Focus on testability

A common refrain from many teams is that testing is difficult (Whittaker, 2000). The source of this perception can come from lack of testing skills and aspects of the software design that make testing challenging.

Whatever the source of the difficulty, when testing is (or is perceived as) a difficult, time consuming process, this creates an affordance for pushing testing activities to a separate testing specialist.

Improving the testability of software can be accomplished in several ways:

Some argue for following guidelines like the SOLID principles as a means to increase testability (Tuck, 2020). For example, the Liskov Substitution Principle and Dependency Inversion Principles both enhance the modularity of software, allowing modules to be replaced or refactored easily, enhancing the evolvability of the system. This same modularity can be exploited to more easily test small units of code, or insert mocks and stubs into the system under test.

Others argue that improving system properties such as observability and controllability directly improves the testability of the system (Marques).

The testability of the system as a whole can also be improved through the addition or improvement of supporting materials, such as testing frameworks, helper methods or test fixture management systems. Outside of direct improvements to the system’s testability, investing in education and coaching around testing skills can also reduce the perceived difficulty of testing.

Whatever approaches are used to improve the testability of the system, improved testability creates an affordance for testing. As testing is perceived as easier and faster, the benefits of software testing can be better understood and realised by the whole team. On a human level, systems that are easier to work with or test also contribute to developer satisfaction (Tarlinder, 2016).

5 Continuous delivery

Continuous Delivery is the practice of developing software such that it can always be delivered to users.

By shifting our focus to continuous delivery rather than simply testing, we are forced to reckon with a different set of challenges. We need to make releasing a safe, repeatable and reliable process. We are forced to invest in automated testing and deployment. We are forced to invest in speeding up the release process with faster feedback cycles.

Humble et al. (2010) identifies 8 Principles of Software Delivery that underpin Continuous Delivery. One of these principles is “Build Quality In”, which advocates for changes to allow easier detection and fixing of bugs, earlier in the process. This principle also rejects the idea of testing as a phase “and certainly not one to begin after the development phase”, and instead argues for all team members taking responsibility

for quality and testing at all times. This principle is supported by removing the QA column as described above.

By making Continuous Delivery a priority, it becomes impossible to defer quality and testing concerns to later in the process. We are forced to create affordances for rapidly delivering quality software.

Humble et al. (2010) also explicitly identifies investing in an automated deployment pipeline “makes it easier to do the right thing than to do the wrong thing, so teams do the right thing.” The practices of Continuous Delivery afford delivering quality software and make it hard to do otherwise.

6 Additional affordances

Consider other ways in which we can build affordances for quality.

Skelton et al. (2019) argues for consciously designing office spaces to support effective collaboration. Another way of framing this, is that instead of mandating more communication, design the physical space in such a way that it affords the communication you desire. If limited communication between developers and stakeholders leads to misinterpreted requirements and rework, moving the desks of the stakeholders closer to the developers is likely to have more of an impact than a sternly worded email. This effectively creates an affordance for improved communication between developers and stakeholders earlier in the process, an approach often referred to as “shifting quality left”.

Building this affordance sends a clear message that delivering quality software is a collaborative process at all stages, not just at predefined “testing” or “review” stages. As the office space is adjusted to afford more collaboration, it becomes easier to involve stakeholders and team members earlier in the process, such as story refinement.

7 Conclusion

Affordances describe a relationship between agents and the world around them that influences and informs their behaviour. They are a powerful way of understanding the designed world around us and the organisations in which we work.

The image of “QA as gatekeeper” forces an adversarial relationship upon many developers and QAs. Shifting our focus from gatekeeping quality to enabling it, requires identifying and creating affordances for quality within our ways of working.

Changing the questions we ask from “how can we improve quality” to “how can we make it easier for everyone to deliver quality” essentially shifts our focus to creating affordances for quality. As we create affordances for quality, we build quality into the process.

References

Beck, K & Andres, C. 2004. *"Extreme Programming Explained: Embrace Change.* Second Edition. Addison-Wesley.

Hendrickson, E. 2002. "Better Testing - Worse Quality?", *International Conference On Software Management & Applications of Software Measurement*, https://testobsessed.com/wp-content/uploads/2011/04/btwq.pdf

Humble, J & Farley, D. 2010. *Continuous Delivery*. Addison-Wesley.

Marques, E. "Testability, controllability, observability". https://moodle-arquivo.ciencias.ulisboa.pt/1415/pluginfile.php/108074/mod_resource/content/1/vvs-slides-13.pdf

Marquet, D. 2012. *Turn the Ship Around!: A True Story of Turning Followers into Leaders*. Penguin Group.

Norman, D. 2013. *The design of everyday things*. Revised and expanded edition. Basic Books.

Poppendieck, M & Poppendieck, T. 2003. *Lean Software Development: An Agile Toolkit*. Addison-Wesley.

Skelton, M & Pais, M. 2019. *Team Topologies*. IT Revolution.

Tarlinder, A. 2016. "Testability from a Developer's Perspective". *InformIT*. https://www.informit.com/articles/article.aspx?p=2730112

Tuck, K. 2020. "Testability: What it is and how to increase it". https://kevintuck.co.uk/testability-what-it-is-and-how-to-increase-it/

Whittaker, J. 2000. "What Is Software Testing? And Why Is It So Hard?". *IEEE Software*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.111.9188&rep=rep1&type=pdf

QA Best Practices:
GUI Test Automation for EDA Software

Ritu Walia, John Casey
ritu_walia@mentor.com, john_casey@mentor.com

Abstract

Many software failures can be traced back to incomplete, inadequate, or poorly-designed quality assurance (QA) testing. Designing and implementing an effective QA process is critical to the delivery of software that performs as intended and delivers the expected results. A graphical user interface (GUI) is often the primary or single point of interaction between the user and the underlying software. No matter how well-planned a GUI and its functionality may be, any failure of the GUI to operate correctly, either from the user's or the owner's perspective, can result in significant loss of market reputation for the entire product. We define and examine ten critical processes in relation to GUI testing and test automation. These include traditional QA testing techniques requiring experience/knowledge on the underlying software and its applications, for effective test execution and software analysis:

- Error identification: determine the common user mistakes likely to be made when using the GUI
- Character risk level: determine which characters may create problems and when/where (e.g., using reserved characters incorrectly)
- Operation usage: determine if/when operations are used incorrectly in the application (e.g., loading an invalid rule file)
- Element relationships: determine if/when different settings or combinations of related elements create problems
- Limitations and boundary values: determine what issues are created when limits are exceeded, or boundary values not observed
- Performance and stress testing: typically observing time and memory consumption performance under extreme conditions
- Smoke testing: finding fundamental instability within builds, to prevent superfluous testing
- Real-world data: using actual data (e.g., customer data) that is not refined or limited, to ensure adequate coverage of customer-critical issues
- Exploratory testing: when bugs are found, performing random testing in the general area, or of elements created by the same developer, to look for additional bugs.
- Efficient bug reporting: giving back a clear bug report that can drive the efficiency of the bug fix

Biography

Ritu Walia: *Software QA/Test Engineer for Siemens Digital Industries Software, Integration's team, EDA tools for IC Physical Verification. Previous positions include Corporate Marketing Engineer and Systems Engineer in different industries. Received an M.Sc. in Electrical and Electronics Engineering, specializing in VLSI and MEMs, from University of Massachusetts, USA.*

John Casey: *Senior Software Quality Assurance Lead for Siemens Digital Industries Software, leading the QA and Testing infrastructure, Integration's team, EDA tools for IC Design Physical Verification, holding 35 years of experience in the Electronic Design Automation industry. Previous positions include Software Engineer, Senior Product Engineer, Senior QA Engineer with leading companies. Received a B.Sc. in Computer Science from the University of Oregon and M.Sc. in Applied Mathematics from the University of Virginia.*

1. Introduction:

Technology in every sector largely revolves around the Integrated Circuit (IC) industry, which is continuously advancing; we can safely say nearly all of us now live in a semiconductor-centric world. The semiconductor industry is historically propelled by a solid correlation between technological scaling and performance improvement of ICs. To support that scaling, the functional complexity of electronic design automation (EDA) tools is constantly expanding as well. In turn, this expansion of EDA technology increases the challenges in the road-mapping processes that include parameters and requirements related to new functionalities.

For advancement in the semiconductor industry, where even a single IC project has a few hundred million dollars at stake, EDA tools are constantly challenged to avail the most efficient and top quality utilities to their users. When quality is of paramount importance, risk of failure must be reduced using every means possible, and productivity is essential. To ensure the most productive tool in hand, testing techniques derived from years of experience play an important role. Various traditional methods not only save significant testing effort, but also contribute towards architecting modern software testing techniques. As we know, the key benefit of quality assurance (QA) is the identification and subsequent elimination of errors in a tool's usage. This process must not only evaluate the reliability of the software behind the tool, but also ensure it operates safely and accurately when used in real-life scenarios. Moreover, the global pandemic accelerated digital transformation, which in turn increased the demand for testing to ensure the best quality outcomes for software and its users.

Traditional QA techniques provide a strong background for the effective implementation of these goals. We demonstrate the applicability of these practices using the tools provided by Calibre RealTime interface from Siemens Digital Industries Software, a platform intended to find errors in the layout of an IC during physical design & implementation processes, enable engineers to check the validity of error fixes applied to that layout, and provide design validation of an IC, prior to delivering the layout for manufacturing.

1.1. Automated, Manual & Exploratory testing

For a tester, it is crucial to understand the application design even as the development code is being written simultaneously. With this understanding, arises the need for an efficient and complete testing plan that must include manual, automated, and exploratory testing:

- Test automation provides more efficient code coverage to ensure all required scenarios are covered for most projects.
- Manual testing saves QA time on smaller projects where test automation might not be needed.
- Exploratory testing may play an important role as a precursor to test automation- for discovery, investigation, and learning new test scenarios.

All of these test strategies contribute to effectively tighten release cycle schedules and minimize the risk of test failure & product performance degradation.

1.1.1. Black-Box Testing:

The implementation of the above-mentioned test strategies broadly encompasses black box testing techniques. Black box testing is an opaque-testing scheme, meaning a tester mutes any working knowledge about the software while testing. This style is advantageous because a tester thinks differently than a developer, and can discover unusual software behavior that a developer might overlook. Black box testing makes a tester analogous to a tourist, whose exploratory senses highlight all the aspects and software functionality from a different perspective. Of course, the difference between a tourist and a tester is that the former just witnesses, while the latter evaluates as well.

Functional, non-functional and regression testing fall into the black box testing category. Black box testing can be performed using existing methods like [2],

- Syntax-driven testing
- Equivalence class partitioning
- Cause-effect graphing
- Boundary value analysis
- Requirement-based testing
- Compatibility testing

1.2. GUI Testing:

The graphical user interface (GUIs) is event-driven and consists of one or more dialog boxes, each of which usually contains multiple controls. The GUI operates on a response to stimuli basis, meaning, the initiation of an event is a response to the click of a button or selection of an option. The user must interactively wait for the task to happen, which differentiates GUI tests from testing of its command line interface (CLI) counterpart, where testers control exactly what happens and when. The interface and user experience play a significant role in the application success when it is released to the market. A GUI testing team pays closer attention to the details of the visual dynamics to ensure customer satisfaction and better usability. They test the various aspects of the user interface, such as visual design, functionality, security, compliance, usability, and performance.

What does GUI testing entail?

Most GUI elements are typically developed using instances of pre-compiled objects stored in a library. The source code of these elements may not always be available for coverage evaluation. Input to a GUI consists of a sequence of events, which are generated when a user interacts with the GUI elements. The number of possible permutations of these events may lead to a large number of GUI states. To ensure adequate testing coverage, a GUI event may need to be tested in a large number of these states. Hence, GUI testing demands validation of every bit of logic, GUI feature, or flow of actions to ensure the application works as expected. Not to forget, this type of testing is also closest to the utility and users' perception of the application, making it extremely valuable.

1.3. GUI Test Automation:

Automation of GUI testing plays an important role by incorporating reusable tests parallel to the development phase, allowing more efficient generation and evaluation of the test results. Automating the tedious and repetitive portions of the task not only reduces costs, but improves the accuracy as well.

Automation of GUI testing also involves the execution of a sequence of instructions that are performed for QA testing, under different conditions (for example, different GUI states). This is considered one of the toughest forms of test automation as it involves setting up a communication interface between the test software and its GUI, which can get very challenging. A complicating factor is that a GUI may be highly subject to change in the form of enhancements and fixes. Such a change drives an increase in the need for repeatability for verification of these tasks. This makes automation an integral part of the testing process. Automation here provides repeatable actions to test functionality as a part of any continuous or agile development process. For example, if a user uses the mouse and keyboard, automated GUI tests would mimic the same behavior by making use of mouse and keyboard clicks or writing to objects present on the user interface, all of which can be incorporated as a part of reusable test script.

When building an automated GUI testing strategy, a tester must first verify the elements of a GUI:

- Validate the font size and readability of font
- Check the alignment of the text

- Check the quality and clarity of images
- Check the alignment of images
- Check the positioning of all GUI elements relative to different screen resolutions
- Verify usability conditions, navigation, data integrity, etc.
- Ensure error messages are displayed accurately [3]

This strategy, if implemented correctly can free up substantial QA resources, both human and computing ones. Automation for some test activities improves the efficiency, and is a necessity for implementing others. While a greater degree of automation can never substitute for a rational, well-organized quality process, considerations of what can and should be automated play an important part in devising and incrementally improving a process that makes the best use of human resources. [5]

The goal is to continue with the process of identifying and deploying automation to best effect, as the organization, process, and available technology evolve.

2. QA challenges in EDA Industry

Some Background:

Design rule checking (*DRC*) is the process engineers use to determine if the physical layout of a chip design complies with the manufacturing requirements, which is defined in the *process design kit* (PDK) provided by the *foundry* [8]. The Calibre RealTime platform, developed by Siemens Digital Industries Software, comprises the Calibre RealTime Custom and Calibre RealTime Digital tools that perform DRC to determine the validity of IC design & physical implementation against the design rules. In our case, a *rule file* contains these design rules.

This platform allows on-demand Calibre sign-off design rule checking in analog and digital design environments during the physical implementation process, enabling engineers working on ICs to optimize their manual DRC fixes and focus on meeting their power, performance and area (PPA) goals in far less time. [4][5][6][7]

Calibre RealTime interfaces apply rules for manufacturing of the circuit layout, such as minimum spacing of each area where the photomasks will be applied to the silicon wafer, to the proposed circuit design, and report any rule violation to the engineer designing that layout. Finding flaws in an IC design & layout prior to manufacturing can save the design company and manufacturer billions of dollars. If the photomasks used to create an IC layout are faulty, then the whole manufacturing process becomes faulty. This means the manufacturing yield (number of working devices made for each silicon wafer) is lower. Lower yields not only waste area on the silicon wafer, but also the time required to test the final circuit layout on each chip.

Therefore, performing quality assurance on EDA tools, Calibre RealTime platform tools in our case, poses various challenges, some of which may be unique based on the type of EDA tool under test:

2.1. Tool integration

The Calibre RealTime interfaces take entry of design data from a specialized editor tool for IC designing/ Placement & Routing (P&R) [9], and display DRC errors in the editor window, both of which require tight integration with the editor. There are many IC design and P&R tools available from different EDA companies. To be competitive, Calibre RealTime interfaces must be tightly integrated with each of these third party tools, both in terms of functionality and performance.

Also, the display of rule violations on the circuit layout canvas must be close to instantaneous. This means, Calibre RealTime tool testing must involve these editors in automated functional, flow, and performance tests. Often, integrating these editors through their testing APIs and driving them through regression testing scripts is challenging due to differences in their overall architecture and the scripting language.

2.2. DRC integration

The Calibre RealTime platform is also integrated with its own circuit DRC tool, so communication must be established between the platform tools, their underlying DRC application, and the external layout editor in regression tests. All of this must happen synchronously for the automated regression tests to be reliable. Because of the existence of so many circuit design rule files from different manufacturers, Calibre RealTime products must have a large comprehensive test suite to cover all of these rule files.

2.3. Data formats

Design data formats differ between the editors for IC designing/P&R, so test-cases must be created for all of these different editor data formats, while keeping them functionally identical.

2.4. Software versioning

While the operating system for running the layout editors and P&R tools is restricted to Linux, each version of Linux supported by the editors (multiple RedHat and SusE versions) must be tested on, and regression tests must be compatible.

Creating regression tests create the following challenges:

- Operating the IC editor and P&R tool with their different forms and testing APIs
- Operating inside the Linux operating system with its different scripting syntaxes and OS levels

2.5. Performance and stress testing

Performance and stress tests must measure the CPU time, the wall clock time, and process memory consumption by both the Calibre RealTime process and the layout editor process. Tools that measure all of these quantities while a substantial circuit design is being processed, must be employed.

Traditional QA testing techniques play a significant role to overcome all of the above challenges. Let's see how.

3. Traditional QA testing techniques

Traditional QA testing techniques incorporate a set of quality attributes that maximize test coverage. If any of these attributes are missing in a functionality, that functionality should be subjected to bug detection and quality analysis.

3.1. Error identification

Error identification is commonly known as the error guessing technique. Inputs are chosen by the tester based on both intuition and experience. While random guessing is acceptable, this technique is most productive when some understanding and analysis is applied to find common mistakes that programmers might introduce into a form or GUI element. This method follows no set rules and is instinctively brought to use. Below are some of the key paradigms of the error guessing technique:

3.1.1. Reserved characters

Testers must understand the underlying software package used to create the GUI, and from that knowledge, determine which characters may introduce problems. For example, with Tcl/Tk programming languages, the characters '$', '[', ']', '{', and '}' are all reserved. Testers should enter these

specific characters when entering information into the form, to determine if the form accurately detects the improper use of a reserved character. Likewise, if the GUI element is written in Java, testers must use the reserved characters in Java when testing data entry in the form.

Testers must also be aware of special use cases involving reserved characters. For example, in the Tcl programming language, a custom command prints out a string of characters enclosed in double quotes. Testers must be aware of reserved characters that can be used within this string as an input. Figure 1 shows results from Calibre RealTime interfaces, when the reserved characters are included in different cases within the enclosed string, which are perceived by the Tcl interpreter differently that resulted in error:

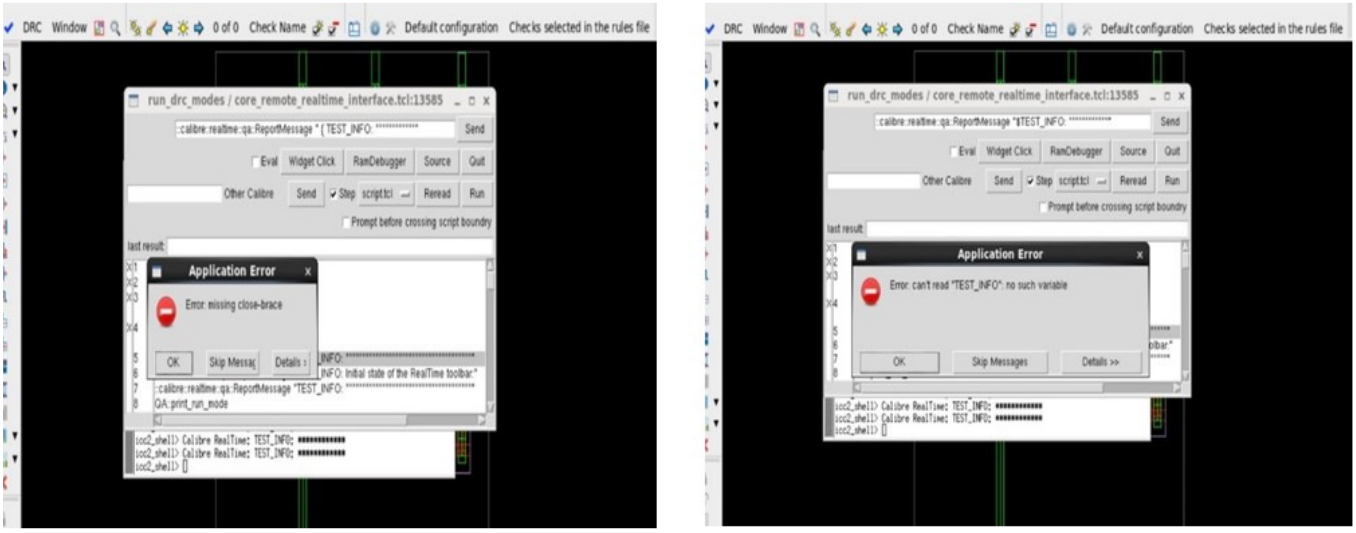


Figure 1. The left screenshot shows the error caused by using ‘ { ’ in the string. The right screenshot shows the error cause by using the ‘$’ sign in the string.

3.1.2. Operation usage

Operation usage is understanding the underlying concept of data under test and determining if/when operations are used incorrectly in the application. Testers must first understand the application of the data being entered. For example, some operations are intended for use on objects of a specific drawing layer of an IC layout design, such as a polygon on layer Metal2. Does the GUI element work when operated on a non-metal layer? What happens if no such element is selected, or if there are no such objects of that type in the design, or if nothing is selected? While programmers in general strive to test their work and understand IC layout design, there may be inexperienced programmers for whom it is necessary to check their work when applied to different data types than what is specified in the original testing requirements. This approach is often more of an “educated guess” as to what might break a certain piece of functionality. Having knowledge of the different types of drawing layers in an IC layout provides testers with the intuition to try different layer types.

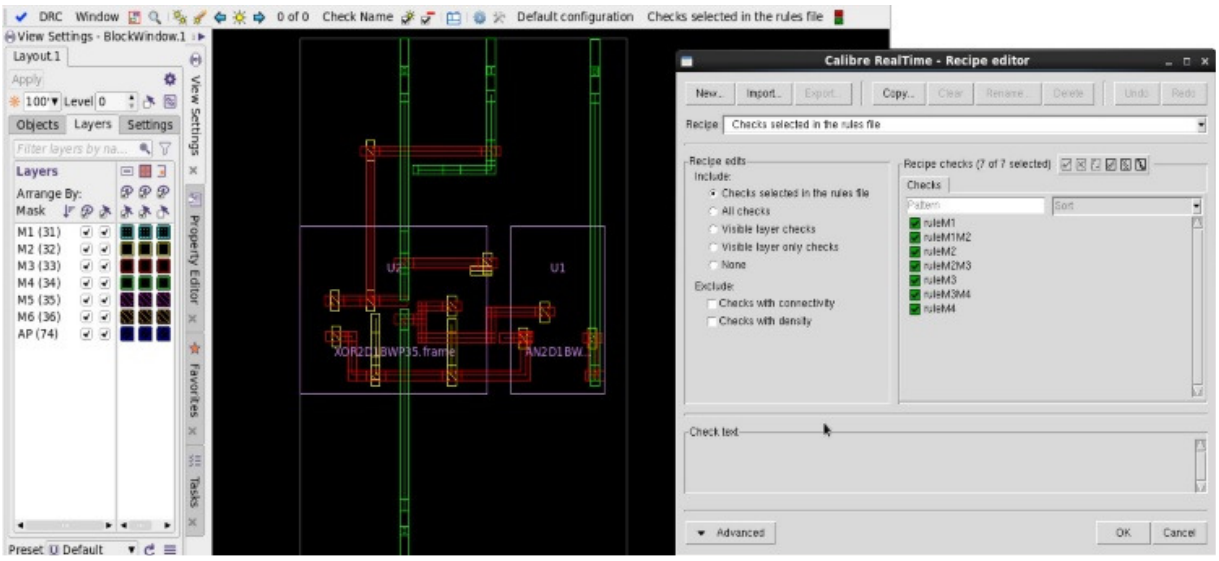


Figure 2. A layer palette from an IC Placement and Routing tool, showing all the layers, with a screen shot of the recipe editor, when clicked on 'Check selected on layer'. The correct operation usage provides tests that involve all layers, not just those such as Metal2 that are involved in routing.

3.1.3. Element relationships and cross-functionality testing

This technique determines if different settings or combinations of related elements create problems. If there are relationships between elements of the GUI, and you are testing one of those elements, do different settings of the related elements result in the correct behavior?

For example, when an IC layout design is sent to Calibre RealTime interface, there is an option to include external files of polygons (in relation to IC layout designing, 'polygons' are used to draw complex orthogonal shapes of metal or polysilicon) to the design, and there is also a polygon limit that determines whether the layout design can be processed at all. Here, it's a good idea to be mindful of potential scenarios such as whether or not the polygons are contained in the external files included in the polygon limit check.

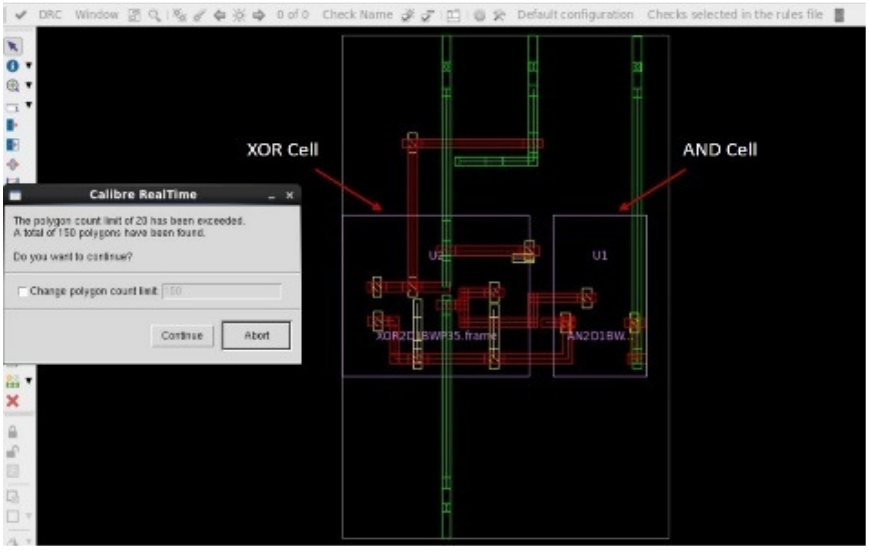


Figure 3. The polygon limit dialog box, along with a layout cell with the XOR cell and AND cell texts pointing to their respective instances. Element relation test case example: Does the polygon limit apply to just the top-level cell, or all cells defined within that layout design?

3.2. Limitations and Boundary Values

The technique of boundary value analysis (BVA) identifies issues that can exist when limits are exceeded, or boundary values not observed. Limitations and boundary value testing is one of the most effective software testing techniques that validates input value ranges, where behavior is expected to change the most. The basis of boundary value analysis is testing the inputs at the boundary limits. It also entails the concept of Equivalence Class Partitioning, meaning that if the input conditions are in the range of x and z values, then a test case should be created with sample data “x-1, x, x+1, y (mid point), z-1, z, z+1” that covers the boundaries (below boundary, boundary value, above boundary) at the two ends of the data set spectrum. Early errors are often found at these extremes. If the program can survive testing at the extremes, it is likely to survive less extreme results.

What kind of data type is being used is handy information in this case. For example, if a developer assigns a variable to be an 8-bit unsigned integer to store input value in the text box, the value of this variable should range between 0-255. Testers must be mindful of cases that may contain out-of-range values (such as an input like -1) or text/input field limitations for strings (where too large a string may cause buffer overflow/data corruption), to detect any unexpected behavior in the application during early stages of testing.

Within the EDA industry, there are often limitations imposed by the design & implementation process or tool being used when entering information into a form. These limitations can usually be found in the process or tool manual. For example, a mask layer number is typically a positive integer in most IC layout tools. Can the entry box of this layer number accept negative numbers, non-integers, or numbers beyond the tool or process limits, when entered? If the entry should be a number, what happens when a common name used in the IC layout design, such as “metal1,” is entered instead of a number? Covering the corner cases is essential for valid limitations and boundary value testing.

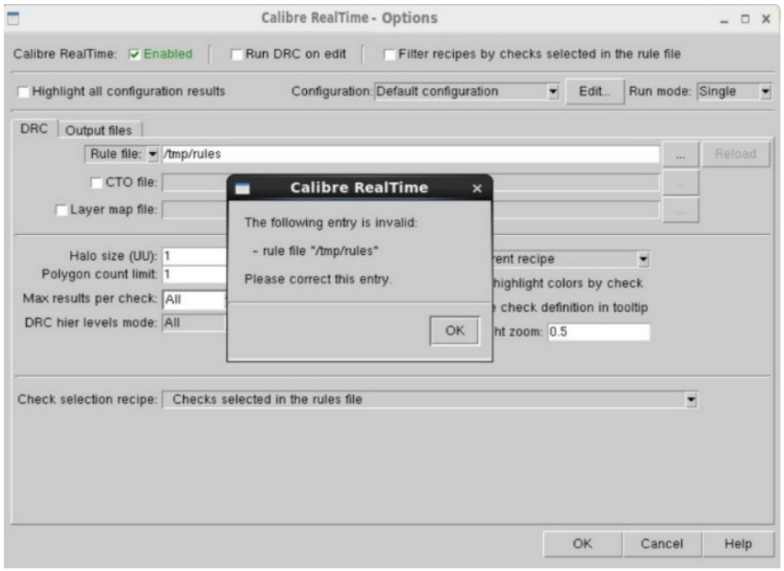


Figure 4: An error dialog when a rule-file with no read permission is uploaded, which is a boundary condition.

Testers however, must take care not to rely on this technique alone, as boundary value analysis and equivalence class partitioning do not explore combinations of input conditions, and may be restricted for an astronomical combination of inputs. For example, each IC layout design can contain hundreds of layers, and hundreds of design rule checks operating on each layer. Different IC layout designs can use different ways to map each layout editor layer to each physical layer, each rule check for each foundry process uses different minimum spacing rules for each layer, etc. Using ‘educated’ guessing of boundary values can expose the most important flaws in the tool, which is vital for test validity and efficiency.

3.3. Functional Performance and Stress Testing

Performance problems in code or functionality that are most critical to customer's success must be identified in performance tests, such as the time needed to identify rule violations and highlight them on the layout editor canvas. Applications must be consistently run and reported on supported hardware so that any degradation in performance can be easily identified and fixed.

Performance tests are usually executed to determine how quickly the program runs to decide whether the optimization is needed or not. These tests can also expose many other bugs. A significant change in performance from a previous release can indicate the effect of an introduced coding error. For example, if testers investigate how long a simple function test takes to run today, and then discover that the same test on the same machine runs much faster or slower tomorrow, they'll probably check with the programmer or investigate for a bug. Either case is suspicious, because something fundamental about the program has changed.

Functional performance tests must include layout designs that are larger, complex, and contain all elements that customers typically have in their layout designs. Asking relevant questions is essential—are forms opening and closing in a timely manner, and are other operations still showing adequate performance at the end of a stress test? Is the memory consumption acceptable, or are there signs of a memory leak?

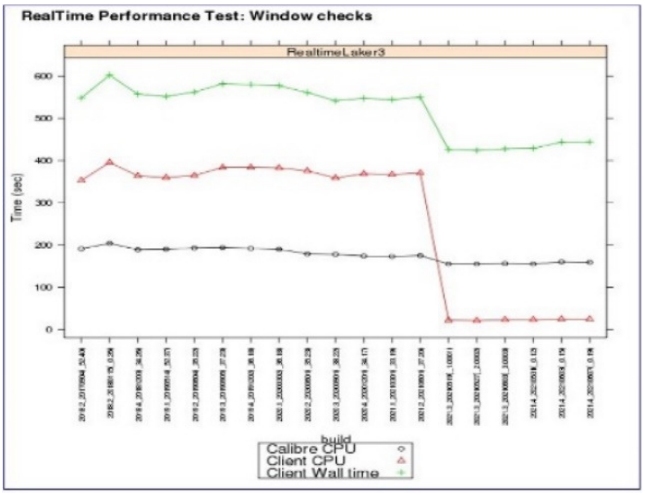


Figure 5. Performance improvement after new enhancements were introduced in the third-party layout editor integration.

Stress tests are also important to test prolonged usage of the application the way a customer would. Stress testing involving multiple designs and multiple layout editors must be performed. For example, testers can simulate a prolonged user session if that is typical customer usage. Another option is to keep inputting information to a form in fast succession, to see if the application can handle this scenario. Since customers of layout editor EDA tools often load their designs and work over prolonged periods of time while a design is in progress, the Calibre RealTime tools must be stress tested over prolonged periods of time as well.

Figure 6 illustrates a design divided up into tiles that has a script performing repetitive DRC runs, then a highlight operation on each tile. The performance would be acceptable if output of the cycle equals the number of DRC checks finished and total memory consumed is within the threshold.

A performance test suite must measure execution/processing time and memory consumption to completely cover all aspects of testing the key features of the application. Underlying third-party API changes may result in significant performance degradation that might be difficult to track, making it highly important to run performance tests as often as feasible. On the other hand, there can be significant performance up-ticks with continuous software improvement that can help the team track progress using the performance charts.

Performance and stress tests are equally important as functional tests. If these test conditions are not covered completely as well, negative customer experiences can occur.

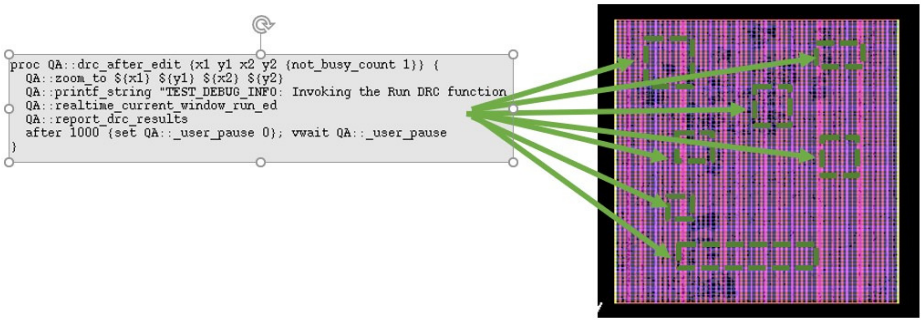


Figure 6. Stress testing: Script that test the software in random areas of a complex design for a prolonged period can uncover performance slowdowns and memory mismanagement

3.4. Smoke Testing

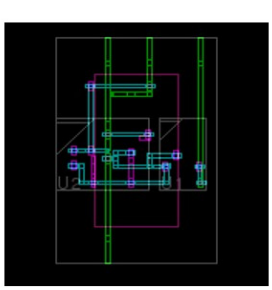
Smoke testing is the build verification testing (BVT), which consists of a standard suite of tests applied to a new build. In testing processes, the aim of smoke testing is to detect major issues early on—the tests look for fundamental instability, or key features that are broken or missing. If these tests fail, tester can abort the testing altogether, knowing that this software build is essentially unstable. Instead, one can continue testing the old build, or wait for the next one. In the simplest terms, smoke testing verifies that the important features are working and there are no showstoppers in the build. It is a rapid mini regression test of major functionality. Smoke tests qualify the build for further formal testing. Smoke tests establish system stability and conformance to the requirements.

3.5. Using Customer Design Data

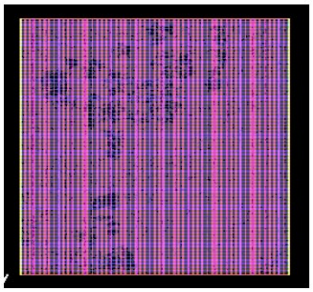
The largest and most complex design flows and data from the customers, should be prominently used in the set of test cases. If the test cases and data are too simple, testers will not cover all the aspects that are important to customers, which can lead to inadequate testing of the product.

When testing the individual functional and performance aspects, it is better to use customer databases as the basis for these tests instead of creating a contrived design. Creating a smaller version of the design is often required to keep the regression run time as low as possible**.**

Most importantly, including customer designs in regression tests may require non-disclosure agreements (NDAs) between a test team and the customer. When required, NDAs should always be completed before testing begins.



Testing a few layout nets, ports, simple devices, 3 layers



Testing many net different types, differently shaped ports, complex macros, 8-9 layers, performance and capacity testing

Figure 7. Testing of simple layout design versus complex customer layout design

3.6. Exploratory Testing

When a problem is found for one GUI item, there is likely to be a problem with related items in it, or the items added by the same developer. Testing that found a bug, can be leveraged to create test cases that have a scope of finding bugs in another GUI element. Example, cross functionality testing between two different windows.

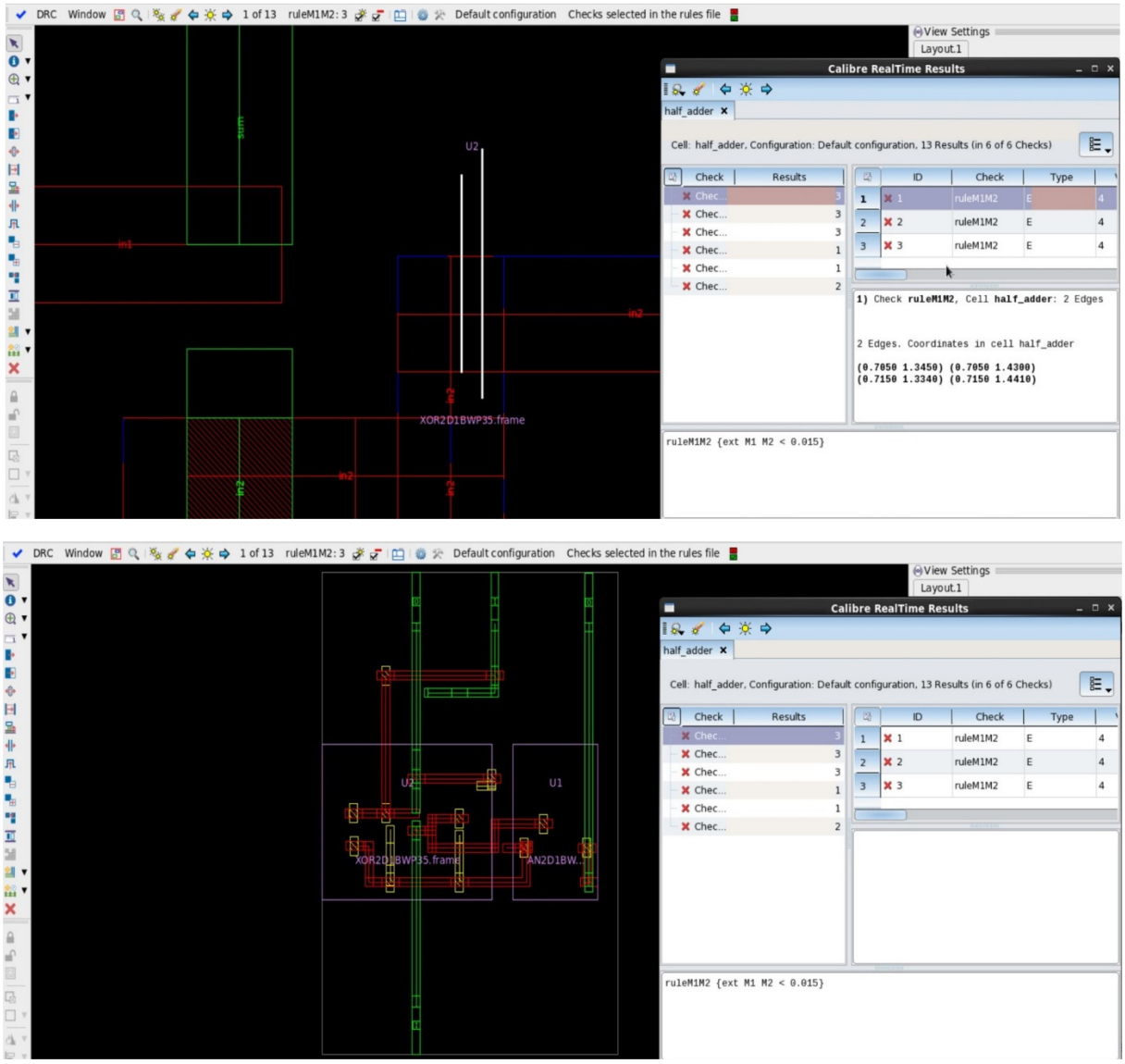


Figure 8.(a)(b). Cross-functionality testing for cross-highlighting between DRC results in the layout viewing window and the Realtime-Results Viewing Environment Window.

3.7. Efficient bug reporting:

Testers should be able to advocate for bug fixes they consider high priority. How testers write and present test outcomes can impact the perception of the reader. An informative, concise, and clear bug report can drive the efficiency of the bug fix, while a weak one may generate extra unnecessary work for the developers. Taking the time to create a value-added bug report can significantly improve overall productivity.

4. Conclusion

The traditional QA techniques described in this paper, when executed correctly, play a significant part in successful development of an IC design & implementation project, and finally to the ROI of the organization.

We discussed the importance of implementing appropriate testing methods for different conditions and operations, and explained how each technique might be evaluated in relation to GUI testing and its automation. We examined the elements of an EDA tool, employed the traditional testing techniques for overcoming QA challenges, and thereby demonstrated the development of an effective quality process. By correctly employing essential techniques, test engineers can ensure the overall quality and productivity of their testing processes. In turn, by providing a thorough and effective QA evaluation, they help ensure that products deliver premium quality and maximum profit when released to market.

References

[1] Bach, James, Bret Pettichord, and Cem Kaner. 2002. Lessons Learned in Software Testing: A Context-Driven Approach. John Wiley & Sons (Accessed May).

[2] Software Engineering | Black Box Testing. GeeksforGeeks, May 2018. https://www.geeksforgeeks.org/software-engineering-black-box-testing/ (Accessed July).

[3] What Is GUI Testing (Graphical User Interface Testing)? Professionalqa.com. May 2019. https://www.professionalqa.com/gui-testing (Accessed June).

[4] Calibre RealTime Digital. Siemens Digital Industries Software. https://www.eda-solutions.com/products/calibre-realtime-digital/ (Accessed April).

[5] Calibre RealTime Custom. Siemens Digital Industries Software. https://www.eda-solutions.com/products/calibre-realtime-custom/ (Accessed April).

[6] Swathi Rangarajan. Tape out on time with on-demand signoff DRC in P&R. Siemens Digital Industries Software. https://resources.sw.siemens.com/en-US/white-paper-tape-out-on-time-with-on-demand-signoff-drc-in-p-and-r (Accessed April).

[7] Srinivas Velivella. Custom & digital layout designers use the Calibre RealTime Platform to close DRC fixes faster. Siemens Digital Industries Software. https://blogs.sw.siemens.com/calibre/2020/12/15/custom-digital-layout-designersuse-the-calibre-realtime-platform-to-close-drc-fixes-faster/ (Accessed April).

[8] Wikipedia. Semiconductor fabrication plant. https://en.wikipedia.org/wiki/Semiconductor_fabrication_plant (Accessed April).

[9] Wikipedia. Place and Route. https://en.wikipedia.org/wiki/Place_and_route (Accessed August).

[10] Memon, Atif M., Mary Lou Soffa, and Martha E. Pollack. 2001. Coverage Criteria for GUI Testing. ACM SIGSOFT Software Engineering Notes 26, no. 5: 256. https://doi.org/10.1145/503271.503244 (Accessed May).

[11] Shilpa Chatterjee Roy. How to Select Correct Test Cases for Automation Testing (and Ultimately Achieve a Positive Automation ROI). Software Testing Help. https://www.softwaretestinghelp.com/manual-to-automation-testing-process-challenges/ (Accessed May).

Quality, Culture, and Process: Coming Together to Find Success

Heather M. Wilcox

Heather.wilcox@nwea.org

Abstract

How does your Development culture affect your quality?

The Quality Control team I work on was recently looking at a series of Data Quality issues that had been found in production. In digging through the "whys" behind the defects, we came to the conclusion that most of the bugs were a direct result of the "Hurry Culture" that the development team had created for itself. From there, it wasn't a difficult leap to the realization that our processes for handling production defects weren't appropriate for a "Hurry Culture". The question then became: Do we alter our culture to inspire a process that creates fewer defects, or do we embrace the "Hurry Culture" and rebuild our bug handling procedures so that teams were more responsive and faster at resolution?

Regardless of the answer to our own question, the lesson at hand is that success is driven by understanding the culture of your Development Department. Once you have that, you can deep dive into whether your tools and processes are appropriate for your development philosophy. This paper will explore the process of examining and diagnosing your team's culture as well as determining whether your development practices, tools, related teams, and other programs adequate support that culture.

Biography

After leaving a potential career in Anthropology, Heather has spent over 25 years working and learning in the software industry, choosing to focus primarily on start-up and small companies. As a result, she has had a broad range of job descriptions which include, but are not limited to: Technical Support Engineer, IS Manager, Technical Writer, QA Engineer, QA Manager, and Configuration Management Engineer. More recently, Heather has moved into a more permanent relationship with a mid-sized educational company where she's been for the last 10 years. This has given Heather a wide range of experiences to draw from in her current role as a Staff Quality Engineer. In her spare time, Heather enjoys fiber arts, equestrian sports, and training donkeys.

1 Introduction

As Quality Engineers, we are constantly searching for better ways to improve our products: new tools, methodologies, and processes. We'll try whatever it takes to build a better mousetrap. However, it's not clear that many of us consider our company or team culture when we're building our quality infrastructure. However, in reality, we should be tailoring our processes and tools to suit the characteristics of our development culture.

The Quality Control team I work on was examining a series of Data Quality issues that had been found in production. In digging through the "whys" behind the defects, we came to the conclusion that most of the bugs were a direct result of the "Hurry Culture" that the development team had created for itself. From there, it wasn't a difficult leap to the realization that our processes for handling production defects weren't appropriate for a "Hurry Culture". The question then became: Do we alter our culture to inspire a process that creates fewer defects, or do we embrace the "Hurry Culture" and rebuild our bug handling procedures so that teams were more responsive and faster at resolution?

Which is the right answer? That is up to you. Either can be right, depending upon the circumstances, your products, and your desired end-state.

The lesson at hand is that success is driven by understanding the culture of your Development Department. Once you have that, you can deep dive into whether your tools and processes are appropriate for your development philosophy.

2 What is Culture

Culture (/ˈkʌltʃər/) is an umbrella term which encompasses the social behavior and norms found in human societies, as well as the knowledge, beliefs, arts, laws, customs, capabilities, and habits of the individuals in these groups. (Tylor, Edward. 1871)

This is to say that Culture describes the way a group chooses to interact both within its membership and externally. It drives the determination of acceptable and unacceptable behaviors as well as expectations and customs. Cultural norms can vary widely and are wholly dependent upon the society in which they are constructed.

A common example of a cultural norm in practical usage is the handshake in modern American society. The handshake has been around since at least the 9th century B.C. and was thought to have begun as a gesture of peace. As a result of its longevity, it exists as a form of greeting in many cultures. Interestingly, even such a simple gesture has specific customary requirements depending on where you are in the world. For instance, the firm "American Style" handshake is considered rude in Turkey and other Middle-Eastern countries. (Wikipedia)

Even within the relatively short span of American history, the handshake ritual has had its own evolution. The Pre-covid modern American process is simple: When meeting someone formally for the first time, the custom is to exchange names and then shake hands. ("Hi, my name is Bob Jones." "I'm Jim Thornton. Good to meet you!" <handshake>) We do this ritual without even thinking about it. However, even this simple exchange isn't the same as it was even 100 years ago. At that point in time, just men shook hands – women were accorded a polite nod, but no actual handshake. But as American culture has evolved, along with the status of women, it became polite to shake hands with women as well. Now we are undergoing a Covid-19 inspired evolution of our formal greeting process. For the sake of disease control, the handshake has all but died. However, a new greeting custom has not yet fully risen to the top. The process of formally meeting new people is now complicated by what to do after the exchange of names. Do we bow? Elbow bump? Knuckle bump? Toe touch? What is the correct answer?

Eventually, a new cultural norm will evolve to replace the handshake. Or perhaps our love of the handshake will win out and it will once again regain its status as "most favored greeting ritual."

Realistically, the handshake tradition is entirely driven by American culture. We could just as easily bow, lock pinkies, wave our hands, or even do nothing at all past verbally acknowledging the other person. But, in the US, we shake hands because that's our cultural norm.

In most societies, Culture drives laws, "acceptable" punishments, societal norms, food choices, intersex interactions, clothing choices…. The list is endless, but the bottom line is that most of the decisions that we make each day are influenced by the culture in which we live.

Within the business world, culture drives how we interact with each other. In Japan the business card is considered a representation of a person's identity. There is an entire process around the giving and receiving of business cards. The success of your business transaction with a Japanese company can be greatly influenced by how you handle a business card given to you by one of their representatives. Whatever you do, don't put that card in your wallet! (Plaza Homes, 2021)

Culture can be both Macro and Micro. A country's culture is clearly a macro concept. When you hear "American Culture", you probably think about football, cheeseburgers, certain TV shows, and other "Typically American" things that most Americans like and are known around the world for enjoying. Micro culture, on the other hand, can be as small as two good friends who have developed their own idioms and rules for interaction. "I saw you were bummin' so I brought you a peach." In this example, someone was unhappy (bummin') and the friend brought them a peach because, in their micro culture, that is an accepted token for inspiring happiness.

In the business world, as in other places, cultures can be nested. Your company may have a macro culture that it has established and nurtures, but different corporate divisions or groups within the divisions can have their own sub (micro) cultures that may or may not be reflective of the greater company culture.

This brings us back to the topic of this paper – figuring out what kind of development culture(s) your team has, how it affects your quality, and what processes you need to support it.

3 Examples of Cultures

As previously described, culture is whatever you make or allow it to be. There are a multitude of Anthropological, Sociological, and Business textbooks that define different types of cultures and describe what they might look like inside of your environment. However, putting a textbook name on your culture isn't as important as determining the characteristics of it and deciding whether that is what you want.

There are as many "culture types" as there are adjectives in the dictionary. These are examples of some of the cultures I've seen or experienced in the engineering world. You may recognize bits of your own world within these examples. Each has advantages and disadvantages depending on your product and the developmental stage of your company.

3.1 The "Hurry Up" Culture

The "Hurry Up" culture values speed over all other things. Teams are rewarded for on-time delivery. Medium and minor defects aren't penalized and are expected. Major and critical defects are highly discouraged but do happen. A company that has intentionally developed a "Hurry Up" culture has mitigation strategies in place to appease customers, troubleshoot defects and to create and release bug fixes very quickly. This kind of culture is a prime candidate for CI/CD. "Hurry Up" teams function under the premise that "Nobody dies if we release a bug. Though the Boss might have a heart attack." There may or may not be documentation and any documentation that does exist may or may not be very complete. In an appropriate situation, such as a Web Development company, the Hurry Up culture can be very successful.

However, if a company is an accidental Hurry-Up (like mine), this can lead to a host of issues. Customer dissatisfaction can be a huge problem since escapes are common and it's likely that there isn't sufficient Marketing or Support infrastructure in place to support them. Additionally, employee attrition is common

since every big escape inspires a frenzy of activity. Without a specialized team or process to deal with critical bugs, teams end up getting constantly jerked out of their regular work plans to handle problems. That kind of continual context switching and pressure eventually leads to frustration, burnout, and employee searches for greener pastures.

3.2 The "Careful" Culture (or NASA Culture)

The old school NASA culture is the exact opposite of the Hurry Up. I use NASA as the example because, if they don't get it right, people do die. NASA-style or Careful cultures use words like accountability, repeatability, safety, specification, and regulation. Everything is carefully specified, documented, and tested. Error rates are measured in the thousandths and millionths of a percent. Failures are a big deal where forensic resources are brought in to analyze and determine cause. Projects are developed using a waterfall methodology and are planned years in advance. Highly regulated industries like Banking, Medical, and Aerospace tend to gravitate towards this cultural style.

A fascinating side note to this example is that, each time NASA has abandoned its Careful Culture, they've experienced terrible results. Both the Challenger and Columbia incidents were determined to have been caused by a shift away from careful to more of a "Hurry Culture", where warnings and concerns from experts within the teams were ignored in favor of "shipping the product", e.g. launching the mission. This is a tragic yet strong example of the value of upholding your chosen culture and the importance of ensuring that your culture is appropriate for your type of business.

Interestingly, the "Careful" culture would mean the end of most technology startups. When you're trying to get that first version out using only the angel money that you sold your soul to get, there's no time for being careful. You assemble the best code you can, and you get it out there as quickly as possible so that people can see it and the income can start flowing. Then you cross your fingers and hope for the best.

3.3 The "Better than Average" Culture

Believers in the "Better than Average" paradigm try hard to take the time to do it right while still shipping in a timely manner. They ship good code on a regular basis and take pride in the quality of their products and the resulting positive reputation. Defects aren't penalized, but minor defects are considered annoying and major bugs are anathema. All defects are taken personally. Programmers in this group say things like, "I can't believe we let that out. I'm so embarrassed right now…"

This kind of culture, or a variant thereof, likely represents a majority of the healthy development teams in the software industry. A "Better than Average" team is usually a collection of honest and hardworking folks who are doing the best that they can to turn out a quality product in a timely manner. Everything about their practices, documentation, and interactions is reasonable and practical. They won't be wowing the world with crazy new technology, but they will impress their customers with solid products that are delivered on a predictable schedule. They are the Peanut Butter and Jelly Sandwich of the software industry.

3.4 The "Innovate or Die" Culture

Startups that survive are often founded with this kind of culture. The healthiest of these have a "push-me-pull-you" type of constant information and idea exchange. Group discussions and problem solving in front of a whiteboard are a daily event. Everyone pushes everyone else to think harder and better. No idea goes un-debated and un-discussed. Innovation is key and every idea is valued. These teams are close, fractious, and are incredibly dedicated to the cause. They work as many hours as it takes to get the job done and they're excited about the outcome. To be clear, this kind of behavior is not sustainable – your employees will eventually burn out and leave if it continues too long. But if you're boot-strapping a company, this kind of culture is a great way to get things off the ground.

3.5 The "Purist" Culture

The Purist-style groups are all about doing it exactly right. These groups tend to be either R&D teams nestled deep inside much larger companies or they are short-lived startups. They'll build their entire product from scratch, including the platform and tools, even if those things have already been built by another company. These folks are purists to the core, who want full control over every_last_detail. They'll take as much time as they can to ensure that the product is exactly the way they want it. It doesn't matter what they're building, but whatever that thing is, it will be beautiful, expensive, and perfect. For obvious reasons, these teams are rarely fiscally successful, but they can be great sources for innovative thoughts, process, and tools. They fill the "think tank" role in the software world.

4 Determining Your Culture

There's an old saying that, if you don't choose a company culture, one will evolve on its own and you may not like what you get. This is a great sentiment to use as a start for digging into your development culture. Does your company have a Vision Statement? Mission Statement? Does everyone in the company know the corporate mission? If the answer is yes, then you've got a place to begin. Hopefully, whatever culture your development team has is a direct derivation of your Company culture. However, that is not always true. It's not uncommon for different divisions in large companies to develop their own sub-cultures which may or may not be aligned with the overall corporate culture. Even individual dev teams can develop their own way of working that is significantly different from the other teams in their department. It's important to determine the culture your team has and how aligned it is with that of your organization.

Sometimes your company is small enough or different enough that there is no "Corporate Vision". Or, you may be launching a new company or a new group. In that case, you get to start from scratch which, although challenging, is really an ideal situation. You get to work with your team to make a conscious decision about exactly what kind of culture you want to have.

4.1 Starting from Scratch

If you are in the unique and privileged position of building a team from the ground up, then you can choose the culture you want to develop. Bring in your entire team. If the members of the team are involved in the creation process and can see their contributions integrated into the final result, they will act as the standard bearers for the organization that you want to build. Their belief in the vision will be passed on to new hires as they are brought into the team. This allows the chosen culture to proliferate and reinforce itself.

A great way to jump start your team culture is by doing a Chartering exercise. This process is detailed in section 5. Chartering provides a methodology for digging deep and figuring out what's important to your organization and how you want to work as a team. The result of a completed chartering is a set of artifacts that you can use to help guide your decisions and ensure that your team stays on track and true to your team and corporate vision.

4.2 Searching the Truth in an Established Team

In the organization where I work, we had several projects where there was a hard deadline and an amount of work to be done that was seemingly impossible. Each time we were challenged and each time, we delivered. "We know you can do it" became a regular refrain from our management. We tossed out good process, attention to detail, and even some testing practices to get things completed in time. When we did succeed in pulling off the impossible, we were rewarded - extra time off, gift cards, all sorts of different appreciations to acknowledge the achievement. Even when big defects were found in the resulting products, quick fixes were assembled, customers were soothed, and the work went on.

Eventually, the culture we had before evolved into what we called the “Hurry Culture”. Clearly, we valued getting it done quickly over all other things, but that came at a cost – big tech debt, lax testing practices, customer dis-satisfaction, and some shoddy process. Ultimately, we didn’t intentionally drive our culture one way or the other, so we ended up with something that we didn’t necessarily want.

Hopefully, you can see from the example how a team culture can evolve on its own simply based on what managers choose to reward and emphasize. Alternatively, if conscious choices are made about what behaviors to encourage and why, team or organizational culture can be driven in a more intentional direction. Culture is like a hot-house flower – it needs constant attention and maintenance to flourish and, without those things, it dies and is replaced by weeds. If you neglect your culture, it can evolve into something ugly and uncomfortable.

This brings us back to determining what kind of culture you’ve cultivated within your workplace.

The search begins with values and rewards. What does your team celebrate and prize? Do you reward getting it done on time or early before all other things? How about releasing a product with such quality that you’ve gone a whole month without a single defect report from the field? (We wish….) Perhaps a high customer satisfaction rating is key for your team? Or maybe it’s all of the above in a “Balanced Culture” where quality, velocity, and customer satisfaction are equally valued. If you can determine the things that are most celebrated around a release, you will discover the core of your team’s culture. Root that central idea or ideas out and then write them down in big letters.

4.3 Put a Name on It

Once you’ve determined what it is that your team values (whether it is intentional or not), put a name on it. Is yours a Customer Culture? A Quality Culture? Are you the “Process People”? Choose a name that epitomizes the thing that your team values most. Use it as a touchstone when it feels like the group is straying from its core mission: “We are Quality!” If you find that your culture resonates with the team and with leadership, then celebrate it. If you have a culture but haven’t done any chartering work, then use your culture as a starting point for those exercises. For example, if yours is a “Customer First” culture, then the word “Customer” should appear in your mission and vision. Your charter should reinforce your culture and the opposite should be true as well since the concepts are closely linked together.

Alternatively, if you determine that your current culture is not what you want it to be, use that name as warning to remind your teams what you’re trying to get away from. Are you a “Hurry Culture” but you don’t want to be? Then, when you charter it’s important to ask periodically, “Is this <mission/vision/value> consistent with who we were (a “Hurry Culture”) or who we want to be? That should help you break the habits and mindsets that developed around your old paradigm and assist you in creating a new culture that is more appropriate for your team.

If you do find that you have an inappropriate culture, put some thought into what you want your new world to be like and give that an aspirational name. “Folks, we are stepping away from the ‘Fast and Dirty’ culture we’ve had since we started up and now, going forward, we will be working towards a ‘Quality First’ environment. We all know that we care deeply about Quality, so let’s show that in everything we do!”

5 Embracing or Rejecting Your Culture

Once you determine what kind of culture you’ve got, you need to decide if that’s what you want and then act accordingly.

5.1 Matching Product to Culture

Even if you have a culture that your team likes, it’s important that the paradigm is appropriate for your product. If you build Heads Up Display software for airplanes, a “fast and dirty” culture is clearly not going to work, since a fail can equate to a plane crash. However, that same paradigm may be just fine for a web development company where customers want to see results fast and are willing to deal with errors.

As you evaluate your culture or begin to create a new one, it's important to consider your product line and account for the requirements of your industry and customers. (Like the need to not die!)

5.2 Loving and Protecting your Discovery

If you've discovered that you have a good, healthy culture that is product appropriate and personifies the things you want your development team to be, or something close to it - wonderful! Your job then becomes maintaining and reinforcing that culture. Directives that come in from sources external to the team should be reviewed to ensure that they're consistent with your desired state. As an example, if you've got a Quality Culture and a request comes in for a project that violates the established standard of quality, that request needs to be halted and redesigned or renegotiated before it is passed to your team. Allowing new work that's not in alignment with your team's core values simultaneously undermines morale and the culture that you've built. However, the rework process can serve as an opportunity to educate other parts of your organization about your team's values and priorities which will help to reduce future occurrences of requirements misalignment.

Empowering your team members is also critical to maintaining your desired paradigm. Ideally, the people in your group believe in your culture. Allowing them to speak up and reject or renegotiate work that's inconsistent with your declared values is vital to maintaining their participation in the process. It also demonstrates a respect and appreciation for their thoughts and feelings. Every time someone speaks up and is unceremoniously overruled because "shipping the code is the most important thing", your team will lose confidence that the whole organization really believes in the culture. It's also incredibly demoralizing and disheartening. If it continues, eventually your team will begin to disengage and to believe that the "company culture" is a joke and that people outside the team are disingenuous and just mouthing the good words for show.

Negotiations in good faith can go a long way to assuaging the quality concerns of your development team and meeting the needs of the business. One of my favorite tactics is providing a "menu" of acceptable options to the business team. E.g. "We can ship the code late with the desired quality, we can ship on time with that feature removed/disabled, or we can ship it now but without the support of the Engineering team." Nobody gets exactly what they want, but everyone gets something and, most importantly, the members of your team feel supported and can see good things happening in real time.

Like anything that is worth doing, maintaining your culture is not easy. It requires vigilance, perseverance, and consistency. Anything less will result in degradation and the loss of the team's confidence.

5.3 Jumpstarting a New Culture - Chartering

It is not unusual for an organization to discover that their culture has morphed into an undesirable state – especially in cases where there was no initial effort to define and pursue a particular paradigm. When the determination is made to create, "fix", or modify a team's culture, Chartering or Re-Chartering is an effective method to reset and restart. Additionally, once you've created a culture and chartered your team, periodic rechartering can be helpful to reenforce your culture or to rejuvenate a team that's gotten stale. Unless the constitution of the team changes drastically or there's some external turbulence that's affecting things, it shouldn't need to be done more than once a year.

I've done many chartering exercises in the last 8 years and I've found that certain pieces are more important than others. I've included brief descriptions of those parts here. A detailed explanation of a complete chartering can be found at: http://www.codegenesys.com/agile-chartering-summary-key-activities-teams-can-use-liftoff/

I need to stop right here and directly thank Diana Larsen for teaching me the chartering process. Much of what I do now is derived directly from her work. It is her article that is linked above.

Before starting your Chartering process, consider what kind of environment you want to create. As discussed before, name your culture, and put that name out where everyone can see it during the chartering process. The most asked question during the session should be, “Does this support and or drive our Culture?” If the answer is “no”, then the statement should be either thrown away or modified so that it does.

5.3.1 Purpose

The “Purpose” portion of chartering is the definition of what the team wants to work on. This is couched within the overall directive provided to the team by a guiding entity. Perhaps a team has been created to build video encoder software. Clearly, that part of their directive is non-negotiable. But the processes and technology behind that code are driven by the team.

5.3.1.1 Vision

The creation of a Vision Statement is the first step in Chartering. This is a description of a future state where the ideal has been achieved: “Software is released easily and with consistently high quality.” The vision should be something obtainable but can also contain an element of “Pie in the Sky” thinking.

It is not uncommon for initial attempts at a Vision Statement to result in pretty a good Mission Statement or two. Be on the lookout for these and, when you see them, set them aside for later. Often, one of these “vision statements” will end up as the actual team Mission.

5.3.1.2 Mission

The Mission is a single statement that concisely describes the “What” and “How” the team will use to achieve their vision. “We will build the best widgets with the highest quality using new technologies.” In this example, the “What” is “The best widgets with the highest quality” and the “How” is “using new technologies”.

5.3.1.2.1 Mission Tests

Mission tests support the Mission statement. They are brief statements that describe the ways in which the team can verify that that are achieving their mission. These should be things which are reasonably actionable and/or measurable. Common examples:

- We explore new technologies as they become available.
- We POC all major ideas before fully executing.
- All code is reviewed by at least 2 developers before check-in is allowed.
- We have a 98% Customer Satisfaction Rating

5.3.2 Alignment

Alignment is the process of aligning all team members around the values that the group feels are important to drive their best work. This portion of the chartering exercise is aimed at how the team will do their day to day work and interact with each other and with entities outside of their team.

5.3.2.1 Shared Values

Shared values are simple, usually single words, that encompass the Values that a team wants to uphold. Common Value words: Honesty, Integrity, Creativity, Innovation, Communication, Fun. Members of the team should all agree that the words are important and there should be a common understanding around what they mean. If there is disagreement around a word, it should either be replaced with a similar Value word that everyone agrees with or thrown out altogether. Ultimately, everyone on the team should agree with and believe in the values.

Creating a common understanding around each value can be achieved through simple rules.

5.3.2.1.1 Simple Rules

Simple rules are exactly what you would expect: Short statements that support each Value. They can be used to prevent misuse of a value. (For example, using the premise of Honesty to embarrass a team member instead of confronting them privately.) They can also be used to clarify how the team intends to personify the value.

Examples:

- Honesty
  - We will always tell each other the truth even if it's inconvenient or uncomfortable.
  - We will always be kind when we tell the truth and will do so in good faith.
  - We will never hold back information.
- Innovation
  - We will always look to the future and Innovate where we can.
  - We will not use a comfortable solution when there is a new and better solution that requires us to learn.

5.3.2.2 Working Agreements

These are statements that define how the team carries out their daily business. They can be serious and/or silly

I find working agreements to be the easiest for the team to create. Often, while trying to complete earlier parts of the chartering exercise, participants will accidently stumble on to working agreements. Try to recognize these and set them aside for this part of the exercise.

Common examples:

- We will not accept half-baked code.
- No meetings before 9 AM
- We will always celebrate our wins and learn from our mistakes.

As part of the creation process, make sure that the working agreements are usable for everybody on the team. My example of "No meetings before 9 AM" only works if everyone on your team is on the same continent and isn't an extreme early bird. Working agreements should also have the same meaning for everyone on the team. "We will not accept half-baked code" may be too imprecise for some teams. Perhaps, "We will only accept code that has been reviewed by at least 2 people." Is a better alternative. Working agreements are only helpful if they can be understood and followed by everyone in the group.

6 Tools for Success

Once you've defined your culture or chartered a new one, it's important to ensure that your tools, processes, and messaging all support it. Without support, it's likely that your culture will eventually mutate into something that co-exists "peacefully" with the structure around it. If you had to charter a new culture to replace an existing and undesirable one, it is recommended that you look at the infrastructure that supports your team. You may find that it's necessary to make changes in your tools and processes to help your newly created culture to survive and thrive.

6.1 Development Tools

Above and beyond the basic compilers, your development tools should support the culture you've chosen. For example, If quality is your absolute top priority, then the majority of your budget needs to be aimed at Quality Tools – Test Automation, AI Testing tools, code coverage and code quality monitors –

anything that helps your QA team do a better job and be more efficient. Of course, these are great tools to have anyway, but if Quality really is your #1, then that's where your tech investment should go.

The same is true for whatever culture you choose. Are you a "speedy delivery" culture? Then you should invest in CI/CD tools, so you can release at any and every moment. If you've got a group of "Purists", save your money for people, since that kind of team is going to want to build their own whenever they can. To be completely trite, "Put your money where your culture is!"

6.2 Processes

As with tools, your team processes should absolutely support your culture. If you've adopted a "Quality over All" paradigm, your process should be well defined and all about cautious releases. You'll also likely have heavier documentation and accountability procedures so that, if something does go wrong, root cause is quickly and easily determined.

In a faster release culture, most processes are going to be lightweight and easy. There should be documentation, because going fast doesn't mean that you can't still do things the right way. But documentation in a speed-based world is likely more automation based and requires less human intervention. It also may be less than ideal, but there should be enough documentation to ensure accountability and a reasonable amount of knowledge transfer.

The same is true for defect handling, code review, and even your processes around bringing work into the team. In a CI/CD based culture, your Marketing team/Product Owners need to feed your team work in small, quickly coded and quickly released stories. In a slower, more careful culture, you might choose a more waterfall style model, where the entire project is planned out ahead of time, with predetermined checkpoints and quality milestones. Both paradigms are completely appropriate for the right kind of product.

6.3 Corporate Alignment - Marketing, Messaging and Support

It's a great thing to have a development team or department that has established and defined a culture that it believes in and is proud of. However, if the rest of your organization isn't on board, success will be harder to achieve regardless of how smoothly things are running within your own group.

When your Marketing plans and Support policies don't align with your culture, this can severely damage your relationship with your customers. If you have a "We won't release until it's right" kind of culture and your Marketing team is promising hard shipping dates to your customers, that's a big setup for failure. Either your development team is going to be upset because a product shipped before it was ready, or your customers are going to be angry because the software shipped late. In each case, it's likely your support resources are going to be hit with a large number of requests – either for information on the product that isn't shipping on time or for help because the product shipped on time but it's broken. If your support team isn't prepared to be inundated in that way, again your customers get disappointed.

If, on the other hand, your teams (and hopefully your entire organization) are aligned, you can set everyone up to succeed regardless of your chosen culture. If you are a "Turn and Burn" shop where speed is valued over higher quality, then your marketing team should be focused on showcasing the amazing speed at which your team can deliver features. Your support services should be set up to handle a high call volume and to communicate easily with customers. Website and social media should be updated frequently with the latest information so that users know what fixes are available when. You want your customers to know and believe that "the solution is coming fast and that's okay!"

In summary, when you choose a development culture, you also need to make sure that the teams and processes that support and are supported by Development are also aligned and prepared to act accordingly.

7 Finding Success

By now, it should be reasonably obvious that there is no one right “Development Culture”. However, it is important that your culture is appropriate for your product and that your processes and the rest of your organization support it. If all those things are in alignment, then you should see some immediate improvements and, eventually, success in many areas.

7.1 Releases

When Corporate culture, Development Culture, and process are all aligned, releases work well and people don’t get surprised or upset. Deployments will always have an element of stress just because of what they are. But your Marketing and Support teams should be prepared and knowledgeable about the event because they are in sync with your Development team, who should be satisfied and confident in the code they are about to ship.

Even on “Turn and Burn” teams, where defects are expected, there is a plan in place. Teams know their code and know where problems are likely to occur. They’re staged and ready for whatever happens on release day. There may be a “SWAT” team, whose entire job is to stomp bugs while the primary dev team continues mainline product work. Again, however, there are no surprises. Everyone knows about and is prepared for the inevitable.

In a “Better than Average” group, Marketing may prep customers for some uncertainty around release times by messaging that, “We’re expecting a May release, but Quality is important to us, so we could ship as late as early June. We’ll keep you in the loop and let you know the second it’s ready.” They’re setting the expectation that there will be a new release soon, but not before it’s as good as it can be. Customers will wait for good software and will expect it to work well when it does ship. The “Better than Average” team will do their best not to let their customers down.

7.2 Quality

Your release quality should be appropriate for your product and culture. Of course, we always want everything to be perfect, but that’s not realistic. What is realistic is a quality level where (again) nobody gets surprised. If your customer finds a bug, you want them to think, “Meh. It’s okay.” It may be “okay” because the customer knows that they’ll get a fix in less than a day or it may be “okay” because it’s a minor defect (the big ones rarely happen) and there is faith that it will get taken care of in a couple of weeks. In either case, the customer isn’t worried because your Marketing messaging is supporting your Development culture and the quality level associated with it. They’re making good decisions around their communications and they are setting customer expectations appropriately.

7.3 Customer Satisfaction

As described above, when your Marketing messaging and decisions are in-line with your Development culture, your customer satisfaction rates should be high. This rating may not necessarily be due to the amazing quality of the software. It may be because defect fixes are easily available faster than the customer can find defects. High satisfaction may also be because customers feel supported. Whenever they call, there’s almost no hold time and the person on the other end is knowledgeable and helpful. Or, perhaps partners are satisfied because they don’t have problems. There is a single, yearly release and, after the upgrade and any configuration changes, things work reliably like they always have.

Regardless of the reasons, a culturally appropriate Support Team isn’t slammed and has the bandwidth to handle customers properly and ensure their satisfaction without having to hound the Dev team for answers. As a direct result, customers feel supported and will be more loyal your product. (Charlton and Ward, 2021)

This may be a set of idealistic scenarios, but when all the parts of a business are aligned culturally, this kind of harmony is not unrealistic.

7.4 Retention rates

Last, and most importantly, when everyone on your team is happy and comfortable, then nobody is looking for a job. If developers and QA aren't feeling pushed to ship code they're not happy with, that is a sign that your Marketing team is aligned with your Development culture. If Support isn't constantly slammed by surprised and disappointed customers, again, Support, Marketing and Dev are in alignment.

This kind of interdepartmental alignment around culture helps to ensure every team - Marketing, Sales, Development, Support, HR, IT– are all on the same page. Marketing can build their PR strategy properly. Sales can make more appropriate promises. Development can deliver an appropriate and expected product, for which a prepared Support team can properly help your customers. With a clearly defined Culture, HR knows how to best help your employees and, more importantly, can accurately target potential new hires that will thrive in your environment. Even IT benefits because they know what to expect in terms of network and resource usage. They have the information they need to plan their work and be successful.

8 Conclusion

Regardless of the kind of culture you have or how you do development, if the entire company embraces the same philosophy around how business should be done and how to function together, then things work smoothly and everyone is happier. It really is that simple. What is not simple is the process of achieving and maintaining that level of alignment. It takes effort, time, and dedication from every team to stay in sync and stay focused on maintaining your culture of choice. However, if you are willing to make the effort, the payoff is certainly worth it.

References

Tylor, Edward. 1871. Primitive Culture. Vol 1. New York: J.P. Putnam's Son (Via Wikipedia: https://en.wikipedia.org/wiki/Culture)

Plaza Homes, “Living in Tokyo” guide, article posted January 6, 2021, https://www.realestate-tokyo.com/living-in-tokyo/japanese-culture/japan-business-card-etiquette/ (Accessed May 25, 2021)

Larsen, Diana, and Nies, Ainsley. 2011: Liftoff: Launching Agile Teams & Projects. Hillsboro: Onyx Neon Press via Citation:

Admin at Codegenesys.com, “Agile Chartering – a summary of key activities teams can use from LiftOff”, article posted September 20, 2016, http://www.codegenesys.com/agile-chartering-summary-key-activities-teams-can-use-liftoff/ (Accessed June 3, 2021)

Charlton, Graham and Ward, Brad, “The importance of Customer Service for Loyalty and Retention”, article posted January 10, 2020. https://www.salecycle.com/blog/strategies/the-importance-of-customer-service-for-loyalty-and-retention/ (Accessed June 24, 2021)

Wikipedia: https://en.wikipedia.org/wiki/Handshake (Accessed July 27, 2021)

Can Adopting a Shift-Left Approach to Testing Shift Left too far?

David D. Winfield

David_Winfield@vanguard.com

Abstract

Adopting a Shift-Left Testing approach is seen as a positive trend in software development. Integrating quality deeper in the software development lifecycle helps reduce costs by finding and fixing bugs earlier in development; however, the notion of shifting left can be misleading, moving the **act** of testing left instead of the **process** of testing. This puts more dependence on developers and may lead to less involvement from the tester, potentially weakening the testing program. Shift-Left Testing may be seen as a test automation-only approach, which could increase this dependency. What elements of the testing lifecycle might be missed, overlooked, or thought of as process "dead weight" if an organization adopts an automation-only approach?

Through use of the V-Model and Testing Pyramid, this paper will attempt to illustrate some parts of Shift-Left Testing that may make a testing program vulnerable. Finding defects is as much about analyzing the software with professional skepticism, subject matter expertise, and understanding the practice of software development as it is finding the defects fast and cheap. Incorporating developers in the testing lifecycle is an improvement in quality and a way to increase collaboration, and this combination is key in developing a more comprehensive testing program.

Biography

Dave Winfield is a Scrum Master and Technical Manager at The Vanguard Group. He has been working in software development and delivery for over 30 years in development, project management. and team management.

Introduction

Software development organizations are always looking for ways to improve the software development process. Over the past 20 years, Agile methods have helped improve software development by getting developers closer to the customer, but another process improvement—Shift-Left Testing—was introduced at about the same time to get testers closer to the developers.

Shift-Left Testing was introduced as a concept[(1)] to show how moving testing from a reporting function to a cooperative effort with development could improve development outcomes. While the definition of Shift-Left Testing was centered on cooperation with developers, a later model continued testing's shift left to user requirements [(2)].

It can be argued that Shift-Left Testing has had the additional benefit of making the developers more responsible for the quality of the code; however, moving the testing left may have the unintended (or, in some situations, intended) consequence of reducing the role of the tester.

Shift-Left Testing

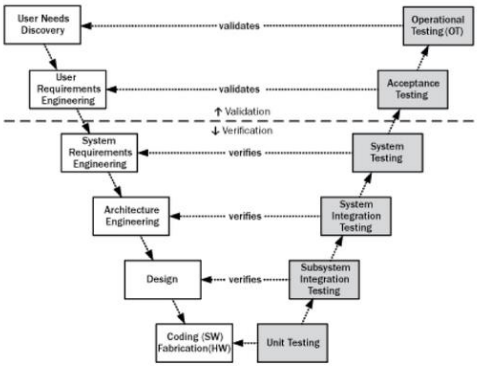
Shift-Left Testing, as introduced in an article by Larry Smith in 2001[(1)], describes "linking" the two functions of testing and development[i]. He wrote that this would reduce costs by finding bugs earlier. In his experience, Quality Assurance (QA) was more a reporting function, producing detailed bug reports for management as a way in gauging quality. His idea was to find the bugs earlier in the process in a cooperative arrangement that would be less about pitting development against testing and more about finding bugs earlier in the code where they would be cheaper to fix.

Smith continues that automation should be a key element of the testing plan. This is vital to his idea that QA should not be running scripts manually but instead these scripts should be automated. Not only was manually running scripts costly to the organization, it was also not a good use of the tester's capabilities—two concepts that continue to be key components in a good testing program. While not all testers would (or could) build automation ("developers are good at developing, testers are good at testing"[(1)]), they can contribute ideas to automation. This article resonates true today where organizations continue to use testers to manually run scripts.

Finally, Smith writes of building "testable code" and automating reporting, and he describes requesting certain log messages to be included in the code he would later analyze. Testable code includes other good software engineering techniques such as making the code easy to maintain and loosely coupled[ii]. By building testable code, it's easier and therefore cheaper to build the automation.

The V-Model and Four Phases of Shift Left Testing

One way to visualize the software development lifecycle is through the V-Model[iii], a graphical representation of the software development lifecycle. This model shows the software development lifecycle in increasing granularity from requirements to coding down the left side of the "V," while each corresponding testing phase is on the opposite side going up the right side. Donald Firesmith, of CMU's Software Engineering Institute, described the implementation of Shift-Left Testing in four distinct models: Traditional, Incremental, Agile/DevOps, and Model-Based[(2)]. The first three "concentrated on beginning testing of the software earlier in the development cycle" [(2)] as described in Shift Left Testing by Larry Smith; however, Firesmith's fourth phase, Model-Based, tries to address testing before software is developed.



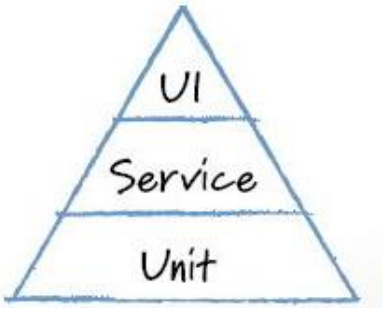
Source: V-Model from Donald Firesmith's Four Types of Shift Left Testing [(2)]

Model-Based Shift-Left Testing looks more deeply at requirements, architecture, and design. Firesmith sees this fourth stage as a goal for those on the Shift-Left Testing journey. In this stage, he says, there is testing applied at all levels of the left side of the V as soon as that stage is complete and before code is written.

The Testing Pyramid

Smith's Shift-Left Testing, as well as the model-based extension proposed by Firesmith, clearly and respectively articulate the need for testing to be early—and even earlier—in the software development lifecycle and to include automation as much as possible. Unfortunately, in the early days of Agile (in this case, Scrum) testing followed the development of executable code and therefore followed in subsequent sprints. Scrum specifies the delivery of completed software, so testing is critical if development is to be complete; however, if it's follow-on testing (i.e., testing the "completed code" in the next Sprint), how can the delivered software be considered complete? Mike Cohn popularized the Testing Pyramid in "Succeeding in Agile" (2009) [(3)] to help Agile development teams conceptualize the quantity of automated

tests they should be applying at each level: Unit, Service, and UI. While there has been some controversy regarding the Testing Pyramid, it follows to put more effort into tests that are easier to write and maintain.



Source: Mike Cohn's Test Pyramid[3]

Test automation was intended to eliminate manually running the test scripts, the most expensive part of testing. The Test Pyramid's depiction was intended to show where the development team should focus their efforts, writing more tests at the Unit level and fewer UI tests (the top layer), as tests at the UI level are more expensive and complex to build and maintain.[4][5]

The Testing Pyramid was introduced as part of Agile methods to help emphasize automating tests, but it's important to remember that with this automation comes development and maintenance—keeping each layer of tests updated with the new requirements and code. Further developments, such as Test Driven Development, are making maintenance of these tests more integral to development which should be less costly.

Shift-Left Testing highlights the need for testers to be more closely involved with specifying the tests even if they don't have the coding skills to implement them. Collaboration is a key component in Shift-Left Testing just as it is in Agile development, but the Test Pyramid does not specify interaction between developers and testers or the tester's role in developing the tests.

Exploring Gaps

In summary, Larry Smith's original article described shift-left testing as collaboration between testing and development, increased test automation, writing testable code, and automated reporting, all to find and fix bugs earlier where they are cheaper[1]. He was clear that testers bring certain talents to software development that developers (not all) are not good at or may not be interested in:

> *"Development programmers are quite simply not good at finding bugs. Nor should they be. Consider the mindset of development programmers: They need to be good coders and good bug fixers—but if they are good at finding bugs, they should be migrating into QA. That is where they can do the most good with such a skill, after all. We should, therefore, not expect development teams to be good at finding bugs. If your management is doing its job, they won't be, almost by definition."*

As Agile methods became more prevalent in the industry, the Testing Pyramid guides teams to increase unit test automation with less emphasis on UI or end-to-end testing automation (while there is value in UI test automation, we value unit testing more). This emphasis, however, can lead to reducing or eliminating the role of the tester on development teams.

"The Four Types of Shift Left Testing", in contrast, move testing even further left (based on the V-Model) to user requirements and user design, while advocating for continued test automation. Firesmith describes the tester and testing in the role of experimentation on the executable code to ensure it operates correctly[iv] and this should include executable requirements.

What happens if an organization thinks it can replace testers with automation at each level? What does an organization lose with this elimination?

Removing Exploratory Testing

A key role for a tester is exploratory testing. Even though automation is a key tenet in Shift-Left Testing, it does not provide for complete coverage of the system. The definition of testing is experimentation[(2)], implying an analysis that can't be performed through scripts. James Bach explains that exploratory testing is "*simultaneous learning, test design, and test execution*"[(6)]. As teams shift left, the importance of exploration becomes critical in the collaboration between developers and testers, as testers can provide additional tests to the developers for automation or input into Test Driven Development as they explore the stories in the Sprint. Removing the tester removes this unbiased critical thinking.

Test Driven Development (TDD) allows the developer to integrate testing in the process of writing code, which keeps the functioning code and the test code in the same workspace. In TDD, developers can't write code without writing tests first; however, these tests are points-in-state and are very granular. As tests are developed, moving up the Testing Pyramid, the tests become less granular, harder to maintain, and represent states of development and functionality further apart. Maintenance of these tests becomes increasingly difficult, so the tester's role is more important to fill in the gaps between test scripts.

Exploratory testing is not random or ad-hoc (as it might be called) but rather reactive to the software being tested. Testers can adjust; while they may not run scripts as precisely as a machine, the tester may notice bugs that would have previously been cast as script error.

Collaboration with Development

Who knows the software system better than the Product Owner? Generally, testers do; they are usually left on the periphery when requirements are written but have unique angles on the software from their experience testing it. New Agile approaches are geared toward automation and are described from the product development perspective more than from software development. A tester's collaboration in product development allows for identification of missing or lacking requirements. Testing can adhere to company standards working with Product Owners to reduce requirements rather than increase them to meet the goals of the customer.

Conclusion

Shift-Left Testing is a key component to an organization's approach to delivering not just bug-free software but also products that continue to delight the customer. Organizations should be aware of the potential pitfalls in removing key components of the testing infrastructure, including the testers themselves. Removing testers from the development teams removes an unbiased perspective and critical thinking. Using a "Model-Based", from Firesmith's paper, approach would incorporate testing at all levels of the SDLC (Software Development Lifecycle) and provide a collaborative relationship between testers, developers, and product owners. Finally, it is important to understand the difference between just running scripts and exploring the software in new ways.

Shift-Left Testing is about shifting the testing program to the left, collaborating with development and product design, and finding bugs before a single line of code is written. Taking Shift-Left Testing too far would be to focus solely on automation, potentially leaving out the tester's unbiased, critical perspective.

References

1.) Larry Smith, https://www.drdobbs.com/shift-left-testing/184404768 (accessed: Aug. 2021)

2.) Donald Firesmith, 2015, SEI Blog, Four Types of Shift Left Testing (cmu.edu) (accessed July 2021)

3.) Mike Cohn, https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid (accessed Sept. 2021)

4.) Alister B Scott, https://alisterbscott.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/

5.) Cem Kaner, 2008, https://www.kaner.com/pdfs/QAIExploring.pdf (accessed June 2021)

6.) James Bach, 2002-2003, http://satisfice.us/articles/et-article.pdf (accessed June 2021)

7.) V-Model (https://en.wikipedia.org/wiki/V-Model_(software_development) (accessed Sept. 2021)

[i] Mr. Smith's paper uses QA and testing interchangeably.
[ii] Like a lot of terms used in software development, there does not seem to be a single definition of testable code but rather an agreement that code written to good engineering principles, like SOLID, will be easier to test and maintain.
[iii] Visually, the V-Model represents the SDLC but with the associated testing phase and distinction between verification and validation. (7)
[iv] Firesmith does a great deal more as he describes the difference between quality assurance and testing, and in fact suggests the use of the term shift-left quality assurance or shift-left verification, which just doesn't have the same ring to it.

Section B:

Select Summaries of Conference Presentations

This page is intentionally left blank.

Build Buy-In--Increase QA's Perceived and REAL Value

Robin F. Goldsmith, JD

President, Go Pro Management, Inc.

robin@gopromanagement.com

Abstract

QA/testing specialists have always encountered resistance. Recently it seems to have gotten worse. We hear of more and more Agile shops eliminating QA/testing specialists in favor of relying on developers to assure their code has suitable quality.

While effective development indeed puts considerable responsibility on developers for creating quality software, history has shown it's necessary but not sufficient. Developers typically are less interested in and less informed about QA/testing concepts and techniques, which reduces their effectiveness. Moreover, it's well-established that there are limits on how well anyone can detect their own mistakes. Surely it should be obvious that independent QA/testing specialists can contribute greatly to creating quality software.

Yet, apparently many managers don't recognize the value of involving QA/testing specialists. This presentation examines common causes of resistance to QA/testing, including seldom-recognized ways well-intentioned but mistaken QA/testing conventional wisdom often backfires, and suggests more effective ways to gain recognition of QA/testing's value—and thereby overcome resistance.

Biography

Robin F. Goldsmith, JD *is one of the few with legitimate credentials in both QA/testing and requirements/business analysis/value, as well as project and process management/measurement/improvement. Uniquely positioned to help you get REAL value right results right, he works directly with project teams assisting, coaching, and training business, systems, and project professionals on risk-based Proactive Software Quality Assurance™ and Proactive Testing™, REAL requirements, project management and leadership, REAL ROI™, metrics, outsourcing, and process improvement. Go Pro Management, Inc. President and SQGNE past President and current Vice President, Robin is author of the Artech House book* Discovering REAL Business Requirements for Software Project Success *and the forthcoming* Cut Creep—Write Right Agile User Stories and Acceptance Tests.

QA specialists are too familiar with resistance to their quality efforts. Resistance can take many forms and stem from many reasons. If resisted QA specialists don't understand accurately why they are being resisted, or if the QA specialists try unsuccessfully to convince others of QA's value and don't accurately understand why, they are likely to continue ineffective practices and even aggravate resistance.

Ordinarily, such resistance comes from others not recognizing QA's value or not relating it to themselves. Conventional wisdom within the QA community often attributes resistance to others not caring enough about quality. Some QA specialists acknowledge that perhaps they have not adequately communicated the value QA provides.

To be fair, a certain amount of resistance to QA is understandable. Many encounter QA primarily in the form of defect reports, which are inherently negative and easily can be interpreted as finger-pointing blame, which nobody likes to have happen to them.

Moreover, developers are largely judged on the basis of whether they deliver working code by their deadlines; and QA often is blamed for delaying delivery past deadlines and thus negatively impacting developers' performance evaluations. Developers often complain they must take added time and effort to comply with bureaucratic and procedural mandates from QA, which developers frequently find mainly to be busywork that seldom adds any real value and actually diverts efforts from activities the developers consider more creative, important, and interesting.

QA specialists seldom recognize how their own behaviors can create resistance and too often equate following QA's procedural busywork mandates with quality. People do change when they can see the What's in It for Me (WIIFM). Just telling people something is good for them isn't sufficient. QA needs to understand the business and development as well as or even better than developer so they can focus on positive quality customers really care about.

Money is the language of business. People have to see the value in money of proposed ideas. Return on Investment (ROI) is a, perhaps *the*, way to show value. However, too many ROI determinations fall prey to common pitfalls that diminish their credibility. REAL ROI™ is a reliable basis for decisions because it avoids those pitfalls.

ROI can be expressed in several different ways. One is payback period. The other is an annualized percentage of net return (benefits) over investment. Such calculations must be analyzed within the context of risk and other factors.

Some key pitfalls to avoid include calculating benefits too narrowly, equating effects on costs and revenue, ignoring timing effects, not quantifying intangibles, justifying costs, and failing to recognize the real source of value. How one measures and communicates QA's value affects how persuasive it is.

Coordinating your manual and automated testing efforts

Joel Montvelisky

joel@practitest.com

Abstract

Due to many historical reasons, most testing and even development organizations, approach their manual and automated testing efforts independently. What's more, when you look closer at these teams, you notice that even within their automation efforts they are using a number of different testing frameworks, running independently and without much thought around coordination, coverage overlaps or functional dependencies.

This approach needs to change. Teams are releasing products faster than ever, and this means that we need to make every testing effort count, including everything from Unit and Integration Tests run by our development teams, Functional and Non Functional automated tests executed by the testing teams, and every manual testing effort encompassing all the Exploratory and Scripted tests run by every member of our teams.

By coordinating the planning, designing, execution, and reporting of our complete testing process we will be able to achieve better visibility and make more accurate decisions faster.

But the road to achieving this goal is not trivial.

This paper will review the objectives of coordinating all your testing efforts, understand common issues and hurdles faced by teams embarking on these efforts and show an approach to coordinate the efforts of your teams in order to generate an orchestrated testing approach

Biography

Joel Montvelisky is a Co-Founder and Chief Solution Architect at PractiTest. Joel started has been in testing and QA since 1997, working as a tester, QA Manager and Director, and a consultant for companies in Israel, the US and the EU. He is a blogger with the QA Intelligence Blog and is constantly imparting webinars on several testing and Quality Related topics.

In addition, Joel is the founder and Chair of the OnlineTestConf, and the co-founder of the State of Testing survey and report.

1 The testing evolution path is pushing towards more automation

We are all witnesses to the ongoing evolution of the testing practices. They've been taking place during these last years and even decades all around us.

There are several factors pushing testing forward, some of them are internal and some external, but all of them point (among other things) to a more extensive and intelligent use of automation and scripting as part of our day-to-day testing tasks.

Among the main factors fueling the evolution of testing we can see:

1. Advances in automation tools - making them more accessible, easier to use, and generating tests are more stable and robust.
2. Changes in the development practices - moving already from Agile towards DevOps and forcing testers to shift our approach to quality and risk in ways that include today concepts such as "testing in production", "chaos engineering" and more.
3. Changes in the "players" - with more teams integrating testers into Agile Teams, at the same time reducing the number of dedicated testers, and all this while making them both more technical and with roles that can be described at times as coaches or internal testing consultants within their own teams.

The overall effect of these factors, together with some others that are not mentioned here, is the increase both in the number of tests being automated, and in the quality and stability of these automated tests and scripts, in comparison to scripts that used to be written in the past.

Having said that, we are also witnessing how manual testing is not losing its importance, especially as teams are being asked to release products faster, amplifying the need for more focused and almost surgical manual testing efforts by the team prior to releasing our products, and before the automated scripts can be developed.

This new reality means that in order to provide an accurate and complete overview of the product from a testing perspective we need to include inputs from both manual and automated testing efforts.

2 Manual and automated tests are completely different in nature

It is not surprising that manual and automated testing are completely different, but we sometimes forget how these differences can make it difficult for team managers to provide a report based on inputs of these 2 sources of information combined.

Not only are manual and automated testing different in the way they are run, but they are also different in the granularity of the coverage they provide with respect to features.

The nature of the issues found is usually influenced by the type of test that uncovered it, as automated tests tend to be more specific and deeper, while manual tests can be a lot less defined and more extensive.

Finally, there is the aspect of execution. While automated scripts may take minutes (maybe hours) to run, making it seamless for them to have many repetitions, manual tests take a considerable amount of time and human effort to execute and will be run maybe a handful of times during a typical release.

If all this wasn't enough, we need to understand that the current trend is for organizations to adopt not one automation framework, but between 3 and 7 frameworks in order to cover all the technological and process related needs of the project. Each of these frameworks will vary with regards to all the parameters we mentioned above, and most times will include its own isolated reporting format.

A final but important remark is that multi-product companies will most times give each team the liberty to choose their own tools and processes, almost promising that the standards and metrics used on their local efforts will be uniquely adapted to the team and its constraints.

This makes the job of the team leads and managers close to impossible, when trying to take the outputs of multiple scattered processes, tools and teams and combine them into a coherent and insightful report to assist your company stakeholders in their decision-making process.

3 Test Orchestration Approach for overall coordination of the testing process

The idea behind the Test Orchestration Approach is to plan and manage the testing efforts in a similar way as a Conductor works with an Orchestra, coordinating the different instruments towards a common result while maximising the value provided by each group and type of instrument.

The approach is based on 3 main phases or effort types:

3.1 Master Planning:

1. Definition of the purpose and scope of the overall testing process
2. Agreement on what needs to be tested manually and what should be tested using automation
3. Breaking down the testing needs into groups and if necessary, into atomic tasks
4. Agreement on priorities and estimated timelines

The purpose here is to have a clear definition of the goals of the testing for the project, as well as an agreement on what should be done by what group and based on what priority level. It is obvious that there will be changes during the actual process but having an agreed baseline will make it many times easier to make decisions on the changes being made to the plans.

3.2 On-Going Coordination:

1. Setting up of a communication and coordination process across teams
2. Deployment of a single reporting and visibility framework
3. Integration between the different tools and frameworks (both manual and automated)

The key to this process is to have coordination, integration and transparency. Coordination can be achieved by setting up a Center of Excellence that will meet and make the cross-team decisions. While the integration and transparency will be achieved by having shared and centralized frameworks that will take the results from each team and make it part of the Quality Dashboards and KPIs.

The above-mentioned Center of Excellence will function as a separate team, meeting a limited numbers of times per month or per period, to review specific topics related your testing process in order to make decision or suggestions for the project teams. It should include senior representatives of your teams as well as external architects and project stakeholders, and they will be able to leverage their experience and knowledge for the benefit of the project.

3.3 Value Broadcasting:

1. Delivering value-adding results both locally to the teams, as well as to the centralized dashboards
2. Ensuring information provides value to your business stakeholders
3. Distribution of information via multiple channels for better reach and diffusion.

This third aspect will ensure that this effort is not a one-time-project, by providing value to the rest of the organization, cementing the place of the testing team as the go-to-team when data-driven decisions need to be made.

4 Understand your specific needs and look for coordinated synergies

The approach and the methodology showed above is generic in nature and it can serve as a guideline to follow in order to implement the Test Orchestration process in your organization.

Still, as it is logical to assume, each organization has their own existing structure and process as well as their own challenges and constraints, and this means that as much as we would like to provide a step-by-step guide to follow this would only result in changes that do not provide the desired results and may even be counterproductive in the long run.

The recommendation is to use the points as a generic blueprint to be adapted to your own needs and constraints, in order to bridge between the specific gaps that exist in your organization.

The most important guiding points on the process remain the same, and those are the ones you should keep in mind throughout the process:

1. Coordination is achieved by considering the needs of all the players and providing a solution that is good for most, even if it is not optimal for any of them.
2. Value comes from information and not from the testing actions you are performing, put as many efforts on distributing the knowledge as you in acquiring it.
3. Ideas and innovation will come from the people in the field, a centralized process should not limit individual teams from experimenting and coming up with improvements that can be shared with the rest of the teams.

Finally, just as software development and testing practices have evolved up to our current place, it is only logical they will continue moving forward. It is important to be critical of our methods in order to not only adopt emerging practices but to help develop them as we take part of this journey.

Reinventing the wheel: Lessons learned from my first year as an Automation Test Developer

Sam Simataa

sam.simataa@gmail.com

Abstract

Transitioning from manual software testing to automation testing seemed straightforward on paper, but as many know - it's easier said than done. Nevertheless, having graduated from college with a technical degree, I decided to take up the challenge.

I was introduced to TestCafe[1], a Node.js end-to-end test automation framework, and spent many hours learning and implementing what I was learning. Finding and analyzing bugs found by the automation framework gave me more insights on how to refine test scripts and what approaches to take when testing User Stories. With more practice, I began to understand the importance of organizing test code and increasing readability by using Page Model patterns. This quality of code can be parsed faster, and problematic sections can be demoed or explained more clearly. I've found this to also aid in nurturing healthier collaborations with other Quality Assurance (QA) team members and even Developers.

Reflecting on my first year in test automation, a profound sense of self-confidence has grown from the lessons learned. Therefore, in this talk I will share a process I've created for myself when learning new technologies that may seem overwhelming: Learn - Implement - Organize - Collaborate - Repeat. This cycle has allowed me to be a more vocal and open team member and continues to fuel my QA passions.

Biography

I am a new Automation Test Developer with little over 1 year under my belt, but with over 7 years exposure to the Information Technology (IT) world. I have mainly worked in the Manufacturing/Construction (Fastenal Company) and Healthcare industries (Preventice Technologies, Mayo Clinic). However, I have also done freelance QA project work through Testlio and tested software in several other industries including Entertainment, Automotive and Finance.

The following are presentations I have done before:

1. Ministry of Testing - UI Automation Week - TestBash().Online 2020
   1.1. Experience Report - Web UI Automation
2. Ministry of Testing - Exploratory Testing Week 2021
   2.1. Experience Report - Shift Left Testing
   2.2. Experience Report - Sharing Your Testing Story
3. PyCon Namibia 2021
   3.1. Quality Over Quantity: An Introduction to Software Testing

As a QA professional, I enjoy getting to know and learn from people of all walks of life and advocating for the user experience. I also enjoy listening to Bach, Drake and A.R. Rahman; playing soccer, tennis and doing triathlons; and travelling, museums and good conversations.

1. Introduction

Having graduated from college with a major in Management Information Systems and minors in Computer Science and Music, I ventured into the world of IT and landed a position as a Support Analyst. Over time, debugging support calls and tickets evoked a sense of user advocacy in me and in time led me to become a Manual Tester. Even though I was satisfied by my passion in advocating for end users and finding bugs, over time I felt the need to dig deeper into QA, and eventually discovered test automation. This felt like a perfect fit: I would still retain my previous QA passions while making use of the technical aspects of my college degree.

I began this journey when I landed a position as an Automation Test Developer at Mayo Clinic in Rochester, MN. One of the chief attractions to them is their primary value: the needs of the patient comes first; this value spoke to my inner value of advocating for the end user. This buy-in would prove instrumental in me keeping the personal motto of “quality over quantity” in the initial days of my test automation journey.

2. Deciding on a Framework

The allure of User Interface (UI) Test Automation is quite powerful, as automation as a discipline has risen in demand. However, getting started can be an inordinate task. This is due to multiple resources and technologies that exist, and the plethora of tutorials as well. It can be overwhelming to start down a path and be exposed to many different learning paths and resources. Fortunately for me, I had a project to think about. Therefore, one way that I was able to overcome this was by creating some criteria to help decide which automation framework and tool would be good to use for the project. One of the decision frameworks I’ve enjoyed using is the Decision Matrix decision making[2] framework, invented by Stuart Pugh. It allows for multidimensional evaluation of several contending options. This made choosing a framework efficient and relevant.

3. Growing in Knowledge

Once I was able to select a test framework, I followed the documentation and examples that were suggested by the TestCafe website first. This allowed me to **learn** and build in best practices as expected by the creators of the technology. In other words, it felt like building up muscle memory to be employed in the future. In my case, using the decision matrix led me to discover TestCafe - a Node.js end-to-end test automation framework. It was easy to set up and get going, and documentation really assisted in building that muscle memory.

Automating the UI portions of User Stories allowed me to practice **implementing** lasting change. I found myself trying to implement tests for a single User Story and would have to refactor when the next User Story came along. It’s as Aristotle once famously said “the more you know, the less you know you know”. However, I also came to find that the more you know, the more you know! It may sound redundant, but my experience supports this in that I was able to refactor code to catch valuable, albeit simple, bugs. Creating these valuable bug reports aided in Gap Analysis of parts that may have been missed in Development or Business Analysis. Just like a piano player

practicing a piece, the implementation coupled with the muscle memory allows for a deeper understanding of what is in front of you.

4. Code Organization and Readability

In an agile world, change is constant, and change is the nemesis of automation. As mentioned above, frustrations came with each User Story, but so did more opportunities to learn. I stumbled upon the concepts of Abstraction and Page Model patterns - allowing code to be **organized** and become reusable components by multiple test files. This allowed me to instill a "Don't Repeat Yourself" (DRY) principle and approach in my test case design. The biggest gain to this was creating a natural sense of code readability. This would come in very handy when trying to demo code or asking for help when stuck.

5. Collaboration

I would like to believe that striving for higher code readability has helped in **collaborations** when either asking for help, trying to succinctly get to the core of an inquiry, or even when trying to propose new changes in the automation code. Creating such feedback loops really helped make me feel supported and competent, even at the beginning level.

6. Conclusion

Now reflecting on my first year in test automation, a profound sense of self-confidence has grown from these lessons learned. I then realized the following process for myself when learning new technologies that may seem overwhelming: Learn - Implement - Organize - Collaborate - Repeat. **Learn** - using resources to choose and understand a technology better, whether this be for test automation or otherwise; **Implement** – adding value to a [work] project by use the gained knowledge to find and develop creative solutions when automating UI portions of a User Story; **Organize** – working smarter and not harder by planning and creating reusable components that can benefit a project in the long run such as using Page Model patterns; **Collaborate** – sharing the ups and downs of your progress in order to garner credibility and create a sense of community and feedback loops; **Repeat** – challenges arise that may be outside the scope of the technology used, so repeating the four previous steps will help overcome some of the challenging barrier. An example could be integrating another technology or moving to a new technology all together.

This cycle has allowed me to be a more vocal and open team member and continues to fuel my QA passions and reminds me that as a QA professional, I should always strive for quality over quantity.

References

[1] *Getting Started*. Docs. (n.d.). https://testcafe.io/documentation/402635/getting-started.

[2] Wikimedia Foundation. (2020, December 16). *Decision-matrix method*. Wikipedia. https://en.wikipedia.org/wiki/Decision-matrix_method.

Driving Postman to meet Newman for REST
The challenges faced, and decisions made

Christina Thalayasingam

niro.tina2k@gmail.com

Abstract

API testing plays an important role in understanding how well our applications can be integrated. There are many API testing tools, frameworks, libraries, and REST Clients out there that help you test APIs. Out of them, all Postman has stood out to be one that has grown into catering to most of the needs that API test. However, no one speaks much about Newman and its goodness remains hidden too many. Newman is a command-line Collection Runner for Postman.

This paper will discuss how you could run Continuous API testing by leveraging Postman and Newman. When coding tools like Super-Test or REST-Assured are not the preferred choices as the team is looking for options that would require less amount of coding. This is the ideal set of tools that make the continuous API Testing a reality.

In this process we will uncover why Postman with Newman are an ideal combination than the other popular tools out there that have similar features. Further the important problems faced with each of the tools that were the case that drove the decision to take the Postman and Newman route. The main focus that leads to the path of Newman and what were the privileges for the team.

This paper will discuss the certain challenges that were faced particularly when multiple API MAUTH-Proxies were required to run successful tests to cover all workflows. Finally, the paper will also discuss hidden goodies that makes Newman a helpful tool.

The paper will conclude with how tools should be selected solely based on the tests necessity and the application that should be tested.

Biography

Christina Thalayasingam has more than 7 years of experience in both functional and non-functional testing. She possesses a development background. Since she has worked on PHP Web Development and Android Mobile Development before taking up Quality Engineering.

She has worked in automate testing content management systems for the UK government, point of sales applications, eCommerce application, and clinical trial applications. She was worked on-site in the UK on projects with the UK government sector and major food supply chain management company.

Christina is currently working as a Test Engineering Manager at Northwestern Mutual, a Fortune 100 financial services company. She manages the testing effort of certain micro apps and services for their Web – Customer Experience Products

Also, she has been part of various prestigious conferences, technical meetups and webinars. She is a software testing evangelist.

Introduction

API is the layer that sits between the user interface (UI) and the database layer in the creation of software applications (apps). APIs allow data and communication to flow from one software system to another.

API testing is a type of software testing that examines APIs directly, including their functionality, dependability, performance, and security. API testing, which is part of integration testing, tests the logic of the build architecture in a short amount of time.

This talk will cover the types of API testing such as Validation Testing, Functional testing, Load testing, Runtime error detection, UI testing, Security testing, Penetration Testing and Fuzz-testing.

This talk covers about how Newman for Postman made the needs of making continuous testing a reality. Many key factors were considered while choosing Newman for Postman over other tools out there. The following factors were considered in deciding which tool fits the purpose. Minimum coding skills require, ability to work with MAuth Proxy and similar Authentication Methods such as OAuth Tokens and AWS Signature, ability to share the scripts and most importantly, incorporating Continuous Testing. Some of these factors are very challenging and

Comparing other tools

Paw, Postman, and Insomnia were some of the most promising tools which fit our search criteria. Here is the comparison that was made.

COMPARISON CATEGORIES	POSTMAN	INSOMNIA	PAW
Authentication	No JWT Generation and AuthO	No JWT Generation and AuthO	No NTLM and others were working via Extensions
Environment	Environments and their variables are segregated per project	Environments can be segregated by workspace – no Variable Grouping	Environments and their variables are segregated per workspace but no color hints
Assertions/Automated Testing	Yes	No	No
Save response as “Example”	Yes	No	No
Plug-ins/Extensions	Does not allow users to create plugin	can create plugins	can create plugins
GraphQL	Yes, includes schema fetching.	Yes, includes schema fetching.	No

Postman with Newman that was fit for purpose

Based on the comparisons that was conducted, Postman was a great fit for most of the criteria and using Newman alongside with Postman made Continuous Testing possible. Newman is a powerful command-line collection runner for Postman. This makes the Postman tests wo work with Continuous testing tools such as Jenkins, Travis without any issues. Newman comes a well packed support of commands as the following so that may of the Continuous Testing needs can be catered for.

COMMANDS	WHAT THEY DO
-h, --help	**Output usage information**
-v, --version	**Output the version number**
--folder [folderName]	**Specify a single folder to run from a collection.**
-e, --environment [file|URL]	**Specify a Postman environment as a JSON [file]**
-d, --iteration-data [file]	**Specify a data file to use either json or csv**
-g, --globals [file]	**Specify a Postman globals file as JSON [file]**
-n, --iteration-count [number]	**Define the number of iterations to run**
--delay-request [number]	**Specify a delay (in ms) between requests [number]**
--timeout-request [number]	**Specify a request timeout (in ms) for a request**
--bail	**Stops the runner when a test case fails**
-k, --insecure	**Disable strict ssl**
-x, --suppress-exit-code	**Continue running tests even after a failure, but exit with code=0**

Newman being a command line tool made it easier to handle MAUTH proxies for Postman scripts to work on different proxies. This was a great support for making the route to continuous testing a reality.

Reference

- https://blog.testproject.io/2021/06/16/api-testing-101/
- https://learning.postman.com/docs/running-collections/using-newman-cli/command-line-integration-with-newman/
- https://learning.postman.com/